A Parsing Algorithm for Iconic Visual Programming Languages

by

Mohammed Ather Ahmed

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

December, 1996

A PARSING ALGORITHM FOR ICONIC VISUAL PROGRAMMING LANGUAGES

BY

MOHAMMED ATHER AHMED

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

Department of Information & Computer Science

December 1996

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA
COLLEGE OF GRADUATE STUDIES

This thesis, written by

MOHAMMED ATHER AHMED

under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Thesis Committee

٩٦/١٢/٢٢
Dr. Muhammad S. Al – Mulhem (Chairman)

Dr. Jarallah S. AlGhamdi (Member)

Muhammad Shafiq
Dr. Muhammad Shafique (Member)

Department Chairman

Dean, College of Graduate Studies

25-12-96
Date

Dedicated to

my

parents,

brothers,

and

sister

whose prayers and patience

led to this accomplishment.

Acknowledgement

I would like to acknowledge King Fahd University of Petroleum and Minerals for all the support extended during this research.

Thanks are due to my thesis committee chairman, Dr. Muhammad S. Al-Mulhem for his advise, guidance, and cooperation. I would also like to thank my thesis committee members, Dr. Jarallah S. AlGhamdi and Dr. Muhammad Shafique for their useful suggestions and cooperation.

Thanks are due to Dr. Muslim Bozyigit for his valuable advise and guidance during my graduate career. I would like also to place on record my appreciation for the cooperation and help provided by all my friends at KFUPM.

Last but not the least I would like to express my sincere thanks and gratitude to my parents, elder brother and his family, sister, and younger brothers, for their help, guidance, cooperation, motivation, and sacrifices.

Contents

Acknowledgement i

List of Tables iv

List of Figures v

Abstract(English) vii

Abstract(Arabic) viii

1 Introduction 1
- 1.1 Need for VPLs . . . 2
- 1.2 Definitions of Visual Programming Languages (VPLs) . . . 3
- 1.3 Classification of Visual Programming Languages . . . 4
- 1.4 Iconic Languages . . . 9
- 1.5 Parsers . . . 10
- 1.6 Problem Definition . . . 12
- 1.7 Overview of the Solution . . . 12
- 1.8 Organization of Remaining Chapters . . . 14

2 Related and Previous Work 15
- 2.1 Syntactic Pattern Recognition . . . 16
  - 2.1.1 Picture Description Language . . . 17
  - 2.1.2 MIRABELLE . . . 19
  - 2.1.3 Picture Processing Grammars . . . 20
  - 2.1.4 Tree Grammars . . . 21
  - 2.1.5 Plex Grammars . . . 23
  - 2.1.6 Web and Graph Grammars . . . 24
  - 2.1.7 ESP^3 . . . 26
- 2.2 Previous Work on VPLs . . . 26
  - 2.2.1 SILICON Compiler . . . 26
  - 2.2.2 Visual Grammars . . . 29

2.2.3 Positional Grammars 31
2.2.4 Picture Layout Grammars (PLGs) 34
2.2.5 Relational Grammars 38

3 Relation Grammars **44**
3.1 Relation Grammars 44
3.2 Relation Grammar/1 58
3.3 1NS-RG 69

4 Modified Relation Grammar/1 (MRG/1) **80**
4.1 Modified Relation Grammar/1 (MRG/1) 80
4.1.1 Language Defined by MRG/1 88
4.2 Comparision of MRG/1 and RG/1 Formalisms 89
4.2.1 Expressiveness of MRG/1 and RG/1 Formalisms 91

5 Parsing Algorithm **96**
5.1 Overview 96
5.1.1 Parsing Method 97
5.2 MRG/1 Based Predictive Parser 100
5.2.1 Details of the Parsing Algorithm 101
5.3 Examples 111
5.4 Time Complexity 126
5.5 Comparision of MRG/1 and RG/1 Based Parsing Algorithms 130
5.6 Parser Correctness 134

6 Comparisions **137**
6.1 Comparisions with Other Formalisms 137
6.2 Comparision with RG/1 138
6.3 Comparision with 1NS-RG 139

7 Conclusions and Future Research **141**
7.1 Conclusions 141
7.2 Future Work 143

Bibliography **144**

Vita **148**

List of Tables

5.1 Comparision of RG/1 and MRG/1 based Parsing Algorithms. 133

6.1 Comparision of MRG/1 based parsing algorithm with parsing algorithms based on other formalisms. 138
6.2 Comparision of RG/1 and MRG/1 - formalisms and parsing algorithms. 139
6.3 Comparision of 1NS-RG and MRG/1 based parsing algorithms. . . . 140

List of Figures

1.1 Visual Programming. 5

2.1 Primitive classes for line drawing of character A. 18
2.2 Directed graph representation of character A. 18
2.3 Production set of tree grammar of example 2.2. 21
2.4 A pattern belonging to the language generated by the tree grammar of example 2.2. 22
2.5 Tree representation of pattern in figure 2.4. 22
2.6 Web grammar productions of example 2.3. 25
2.7 A sentence belonging to the language generated by web grammar of example 2.3. 25
2.8 An iconic sentence. 28
2.9 Visual grammar productions for a family of barcharts. 30
2.10 A sample visual sentence. 34
2.11 Phases of a PLG based parser. 37
2.12 An expression in L(FRG1). 42

3.1 Parse tree for sentence w. 50
3.2 States and Transitions in a statechart. 53
3.3 Parsing algorithm based on RG/1. 65
3.4 A sample generated by 1NS-RG of example 3.6. 71
3.5 Parsing algorithm based on 1NS-RG. 75

4.1 A sample iconic sentence. 82

5.1 Organization of lexical and syntax analysis phases. 97
5.2 Overview of the Parsing Method. 99
5.3 MRG/1 based Predictive Parser. 100
5.4 Predictive parsing algorithm for Iconic VPLs based on MRG/1. . . . 103
5.5 Parsing algorithm for MRG/1 contd.... 104
5.6 Parsing algorithm for MRG/1 contd.... 105
5.7 Flowchart for MRG/1 Predictive Parser. 106
5.8 Flowchart for MRG/1 Predictive Parser contd... 107

5.9 Flowchart for MRG/1 Predictive Parser contd... 108
5.10 Sample iconic sentence reproduced from Figure 4.1. 111

THESIS ABSTRACT

Name: MOHAMMED ATHER AHMED

Title: A PARSING ALGORITHM FOR ICONIC VISUAL PROGRAMMING LANGUAGES

Degree: MASTER OF SCIENCE

Major Field: INFORMATION & COMPUTER SCIENCE

Date of Degree: DECEMBER 1996

Learning programming is a difficult and time consuming task. To simplify programming, different programming paradigms such as imperative, functional, logic based, object-oriented have been developed. The languages based on all these paradigms have a common factor - the medium of expression, which is textual. Another dimension for the development of programming languages is to change the medium of expression from textual to visual. Visual Programming Languages (VPLs for short) are aimed at doing it. To put VPLs on a par with Textual Languages, lot of work needs to be done on grammatical formalisms, parsing techniques, and compiler generation tools. This research is a step in that direction.

It analyzes various existing grammatical formalisms and parsing techniques for a subclass of VPLs called Iconic Languages. It also proposes a new grammatical formalism called Modified Relation Grammar/1 (MRG/1 for short) to model Iconic Languages, and a predictive parsing algorithm based on it. MRG/1 is compared with some of the existing formalisms. It is shown that MRG/1 is a clear way of expressing VPL syntax; and parsing algorithm based on it is more efficient.

Keywords: Programming Languages, Textual Languages, Visual Programming Languages, Iconic Languages, Syntax, Semantics, Grammatical Formalisms, Parsers, Lexical Analyzers, Time Complexity.

King Fahd University of Petroleum and Minerals, Dhahran.

December 1996

خلاصة الرسالة

اسم الطالب الكامل : محمد اطهر احمد

عنوان الدراسة : خوارزم لاعراب لغات البرمجة الصورية

التخصص : معلومات وعلوم الحاسب الآلي

تاريخ الشهادة : ديسمبر ١٩٩٦ م

تعلم البرمجة مهمة صعبة وتحتاج الى وقت. و لتسهيل البرمجة طورت نماذج مختلفة منها الأمرية والوظائفية والمنطقية والكائنية. وتشترك جميع هذه النماذج بأنها نصية. وهناك اتجاه آخر لتطوير لغات البرمجة وهو التحول من النصوص الى الصور. و يطلق على هذه اللغات لغات البرمجة الصورية. ان الكثير من الجهد مطلوب لصياغة القواعد وطرق الاعراب وأدوات تكوين الترجمة لهذا النوع من اللغات، ويعد هذا البحث خطوة في هذا الاتجاه.

في هذا البحث تم تحليل مجموعة من قواعد اللغات الموجودة وطرق الاعراب لنوع من اللغات الصورية يسمى اللغات الايقونية، ويطرح هذا البحث طريقة جديدة لصياغة القواعد تسمى قاعدة العلاقات المطورة/١ لتمثيل اللغات الايقونية وكذلك خوارزم للاعراب معتمدا عليها. و تمت المقارنة بينها وبين القواعد الموجودة وأظهرت وضوح هذه القاعدة وفعالية طريقة الاعراب.

مفاتيح : لغات البرمجة، اللغات النصية، لغات البرمجة الصورية،اللغات الايقونية، المبنى اللغوي، المعنى، القواعد، الاعراب،محللات اللغة، تعقيد الوقت .

جامعة الملك فهد للبترول والمعادن

الظهران ، المملكة العربية السعودية

ديسمبر ١٩٩٦ م

Chapter 1

Introduction

Learning programming is a difficult and time consuming task. Writing good programs (efficient, modular, having good interface, documented etc.) is even more difficult. To simplify programming and to make better use of human capabilities different programming paradigms such as imperative programming, functional programming, logic programming and object-oriented programming have been developed. In all the programming languages based on these paradigms there is a common factor - the medium of expression. All these languages are textual, i.e., they use textual sentences to express computations. Another dimension for development of programming languages is to change the medium of expression. Visual Programming Languages (VPLs for short) are an attempt to do it, where the medium of expression is graphical rather than textual.

A visual programming language is a programming language which uses a combination of text and/or graphical symbols to express computations. Research in VPLs has mainly focused on development of visual programming languages for specific applications. However in mid 80s some work was done in the area of grammatical

formalisms, and syntactic and semantic analysis of Visual Programming Languages. To put VPLs on the same platform as Textual Languages lot more needs to be done in this area.

1.1 Need for VPLs

One frequently asked question is, despite the availability of so many programming paradigms and programming languages, what is the need of VPLs. This question can be answered by looking at the current status of end user software, and the vast user community and their needs.

Dramatic decrease in hardware costs have made computers cheap and widely available. In fact in U.S. there are more computers than people. Obviously all the computer users are not computer professionals. To make life easy for computer users GUIs and other accessories such as Word Processors, Spreadsheet Programs, Application Generators etc. have been developed. But according to a survey conducted by Rochart and Flannery of Sloan School of Management [12] about 50% of end user applications are not in these predesigned categories. To simplify programming of such applications VPLs have been suggested.

1.2 Definitions of Visual Programming Languages (VPLs)

Numerous definitions of Visual Programming Languages are given in the literature. Some of the important ones are given below:

Nan Shu [32] states that "A Visual Programming Language can be informally defined as a language which uses some visual representations (in addition to or in place of words and numbers) to accomplish what would otherwise have to be written in traditional one-dimensional programming language." This represents a conceptual departure from traditional programming approach and is stimulated by the following premises:

1. Pictures are more powerful than words in terms of communication.
2. Pictures are easy to understand and remember.
3. Pictures do not have language barriers.

S.K.Chang in [27] says that "Visual Programming Languages, usually deals with objects that do not have an inherent visual representation. This includes traditional datatypes such as arrays, stacks, and queues and application datatypes such as forms, documents and databases. Presenting these objects visually is helpful to the user."

Our definition of Visual Programming Languages is as follows:

Definition 1 *Visual Programming Languages are languages which use combination of text and meaningful graphical symbols to write programs. These graphical/textual symbols represent the following:*

a. Datatypes such as numbers, characters, pictures and sound.

b. Data structures such as arrays, records and pointers.

c. Programming constructs such as sequence, selection, and iteration.

1.3 Classification of Visual Programming Languages

The term Visual Programming Languages has been used in many different ways. Often visualization, languages for visual interaction, languages for manipulating visual information, are thought of as Visual Programming Languages. The common thing about all these languages is that they all deal with both graphics and computer programming. To make clear distinctions among them Shu [32] has suggested the classification[1] in Figure 1.1.

As it is evident from the figure, we have two categories: Visual Environments, and Visual Languages. In Visual Environments graphical techniques and pointing devices are used to provide environments for program construction and execution, for information retreival and presentation, and for software development. Whereas Visual Languages are designed to handle visual information, support visual interaction, and develop programs using visual objects. As shown in the figure these categories

[1]We will stick to this classification throughout our work

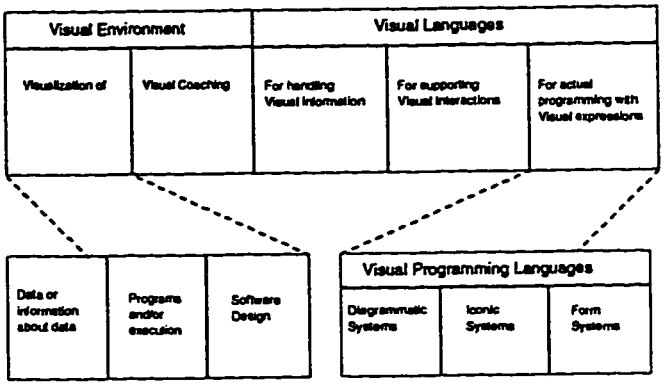

Figure 1.1: Visual Programming.

are further split into subcategories. A brief description of each of the subcategories is given below:

Visualization of Data or Information: Data is stored internally in traditional databases but presented to the user in graphical form. User can use the graphical form to navigate through the databases. This technique is called Spatial Data Management System (SDMS for short). It is motivated by growing database user population who are not computer professionals or experts in use of databases. A prototype SDMS which interfaces to conventional DBMS was built at Computer Corporation of America [11].

Visualization of Program and/or Execution: These systems/tools helps in program construction, debugging and testing. Structured templates are provided for building programs. Options are provided for displaying runtime behaviour of programs, upon their execution.

PECAN [26] is an example in this category. Its current version is based on Pascal and is designed for algebraic programming languages. It includes structured templates for building the program, provides immediate feedback on semantic and syntactic errors while the user is editing, provides 'undo' and 'redo' facilities, and supports various views of program execution.

Visualization of Software Design: These systems are collection of tools which presents requirements, specifications, design and system structures in graphical form to software developers, users and maintenance personnel. In 1981 Computer Corporation of America launched a project to develop such a Program Visualization (PV) system. After carrying out 6 month survey their team recommended the following ten categories of tools:

- System requirements diagrams.
- Program function diagrams.
- Program structure diagrams.
- Communication protocol diagrams.
- Composed and typeset program text.
- Program comments and commentaries.
- Diagrams of flow of control.
- Diagrams of structured data.
- Diagrams of persistent data.

- Diagrams of the program in the host environment.

A prototype PV system developed by Brown et al. [2] supports some of these.

Visual Coaching: In a visual coaching environment user demonstrates steps of computation using sample data values. These data values could be just (input, output) pairs or detailed traces of a program. After this, a program is developed by the visual coaching environment which performs the same transformations on subsequent data values. Pygmalion developed by Smith as part of his Ph.D. dissertation is an example [5].

Languages for Handling Visual Information: These are Textual Languages for manipulation and querying of visual data. GRAIN [32] (Graphics Oriented Relational Algebraic Interpreter) falls in this category. It was incorporated in DIMAP, a system for interactive map data retrieval and manipulation with zooming facility in a distributed database environment. The aim of GRAIN/DIMAP system is to provide techniques for flexible and efficient retrieval of data from pictorial databases.

The main idea in [32] is to represent pictures as logical and physical pictures. A physical picture is stored as an image in an image store. A logical picture, which is a description of physical picture, is stored in relational database tables and manipulated by data manipulation language. Once a logical picture has been identified for retrieval the corresponding physical picture is generated on the display device by retrieving bit pattern from image store.

Languages for Supporting Visual Interaction: These are Textual Languages designed to define, create, and manipulate pictorial symbols. HI-VISUAL [20] developed at Hiroshima University in Japan is an example of this class. It is based on "hierarchical multiple window" model. Desks (virtual displays), region frames, and view ports are the basic elements of the model. The language provides operations for changing the size, location, and shape of frames or scenes. Statements are also provided for associating objects or procedures with the image or desk.

Visual Programming Languages: These are languages for programming using graphical expressions. Shu [32] has further classified them, based on principles of design, as Diagram Based Languages, Iconic Languages and Form Based Languages. A brief description of each of these categories is given below

Diagram Based Languages: Flowcharts or other diagrammatic representations are incorporated into programming languages or made into machine interpretable units to be used along with conventional programming languages. PIGS [3] is an example of Diagram Based Languages.

Iconic Languages: Icons are deliberately designed for programming. An icon could denote a datatype, operator, control construct, procedure name, and so on. User combines these predefined icons to write programs. Pict system [25] developed at the university of Washington is an example of Iconic Languages.

Form Based Languages: In this case programs are written by filling in some forms. It takes an automatic programming approach rather than an algorithmic one. These languages are used in data processing applications. FORMAL System [4] falls in this category.

1.4 Iconic Languages

Since our work applies to Iconic Languages, let us discuss this category in some more detail. In Iconic Languages, user is provided with a library of predefined icons. These icons could denote datatypes, data structures, operators, control constructs, procedures, and so on. Each icon has a distinct shape. The color and size of an icon do not carry any syntactic or semantic information [28]. But we believe that the size or color may carry semantic information but not syntactic information. This is because an icon in Iconic Languages is treated as a token. Syntactic specification of a language describe valid combinations of tokens. The attributes of tokens (viz color and size of icons) are not taken into consideration in syntactic specification. But these attributes may carry some semantic information.

Programming is performed by combining icons according to the syntax of the language. Usually horizontal and vertical adjacency or connection, enclosure, and overlapping, are the permitted compositions (operations). Ofcourse the exact nature of compositions allowed are language dependent.

As Shu [32] puts it, "Iconic Languages are rich in the extent of visual expression". By visual expressions she means meaningful visual (non-textual) notations used as language components in programming. Extent of visual expressions is a relative measure of how much visual expressions are incorporated in a programming lan-

guage. Usually Iconic Languages are application specific and have a limited scope. But still, as Chang [8] puts it, "Iconic Languages continue to be of interest to researchers in Visual Languages community". The formal specification and parsing of Iconic Languages have been studied by many researchers. Grammatical formalisms such as Relation Grammars, Picture Layout Grammars, and Positional Grammars have been proposed to model them. But lot more needs to be done on syntactic and semantic aspects of Iconic Languages to bring them on a par with Textual Languages.

1.5 Parsers

A parser for a grammar G is a program that takes as input a string w and produces as output either a parse tree for w, if w is a sentence of L(G); or an error message if w is not a sentence of L(G). L(G) denotes the language generated or described by grammar G. Often the parse tree is produced only in a figurative sense; in reality the parse tree exists only as a sequence of actions made by stepping through the tree construction process [1]. Numerous parsing algorithms have been developed both for textual and multidimensional (visual) languages. Some parsing algorithms such as Earley's algorithm and CYK algorithm are general and are applicable to both Textual and Visual Languages. The drawback of these algorithms is that they are very inefficient.

For Textual Languages Context Free Grammars (CFGs for short) are widely used as grammatical formalisms. Parsers for CFGs are of two types: top down and bottom up. Bottom up parsers build parse trees from the bottom (leaves) to the top (root), while top down parsers do exactly the opposite. The same classification can be

extended to parsers for Visual Languages also.

Bottom up parsing methods are applicable to wide range of languages and are also very efficient. A class of bottom up parsers called LR parsers are widely used in many compilers. L denotes left to right scanning of input and R denotes rightmost derivation in reverse. Hence LR parsing is basically a reverse of rightmost derivation. LR parser and likewise any bottom up parser is difficult to implement by hand. One needs parser generator tools to implement them. Since such tools are not widely available for Visual Programming Languages we will focus our attention on top down parsing.

As we have already seen top down parsers build the parse tree from root to the leaves. This can be viewed as an attempt to find a leftmost derivation for an input string. Although they are easy to implement, top down parsers have the following disadvantages:

- If the underlying grammar is left recursive then all the left recursion must be removed before applying top down parsing on it.
- The second problem concerns backtracking. The parser makes repeated scans over the input and tries various alternatives (alternate productions) while parsing the input. Practically this involves undoing the semantic effects caused due to erroneous expansions (alternate expansions). This causes substantial overhead, as a result of which top down parsers are inefficient. Top down parsers with no backtracking are usually preferable.
- The order in which alternates are tried effects the language accepted.
- On detection of error we have very little idea of its location.

In many practical cases, top down parsers need no backtracking; proper alternate is selected by just looking at the first symbol it derives. A nonbacktracking top down parser is called a predictive parser. A predictive parser can be implemented in two ways

1. As a set of recursive procedures, in which case it is called recursive descent parser, or
2. In a nonrecursive manner by maintaining an explicit stack of grammar symbols. In this case, the parser has four parts input buffer, stack of grammar symbols, parse table, and parsing program.

1.6 Problem Definition

The purpose of this work is threefold

1. To analyze existing grammatical formalisms and parsing techniques for Iconic Languages.
2. To propose a new grammatical formalism and parsing algorithm for Iconic Languages.
3. To compare our grammatical formalism and parsing algorithm with some of the existing ones.

1.7 Overview of the Solution

In order to meet the objectives specified in the previous section we proceeded as follows

1. We analyzed some of the existing formalisms for VPLs such as Visual Grammars, Picture Layout Grammars, Positional Grammars, Relational Grammars, and Relation Grammars. Also surveyed some of the formalisms used in syntactic pattern recognition such as String Grammars, Picture Processing Grammars, Tree Grammars, Plex Grammars, Web and Graph Grammars. The parsing techniques based on these formalisms are also analyzed. A critical review of all these formalisms and parsing techniques is given in the next two chapters.

2. A new grammatical formalism (basically a variant of Relation Grammar/1) called Modified Relation Grammar/1 (MRG/1 for short), to model Iconic Languages, is developed. Some iconic sentences are modelled using this formalism. MRG/1 formalism is compared with RG/1 formalism. A predictive parsing algorithm based on MRG/1 is proposed. The algorithm is applied to example sentences and steps are traced. Time complexity of the parsing algorithm is derived. MRG/1 based parsing algorithm is compared with RG/1 based parsing algorithm. Proof of correctness of MRG/1 based parsing algorithm is also given.

3. MRG/1 is compared with Picture Processing Grammars, Visual Grammars, Picture Layout Grammars, and Positional Grammars. MRG/1 based parsing algorithm is shown to be more efficient than parsing algorithms based on these formalisms. A detailed comparision of MRG/1 with RG/1 and 1NS-RG is performed. It is shown that MRG/1 is a clear way of describing Iconic VPLs and parsing algorithm based on it is more efficient.

1.8 Organization of Remaining Chapters

Chapter 2 gives a compendium of related and previous work, concentrating on VPLs syntax specification and parsing techniques. Research from Syntactic Pattern Recognition on specification and recognition of pictures using grammars is also reviewed. Chapter 3 describes RG formalism and its restricted forms viz RG/1 and 1NS-RG in detail with examples. Chapter 4 describes MRG/1 in detail with examples. Comparision of RG/1 and MRG/1 as grammatical formalisms is made. Chapter 5 presents parsing algorithm based on MRG/1. The parsing algorithm is applied to example iconic sentences and steps are traced. Time complexity analysis of parsing algorithm is performed. MRG/1 based parsing algorithm is compared with RG/1 based parsing algorithm. Proof of correctness of MRG/1 based parsing algorithm is presented. In chapter 6 the efficiency of MRG/1 based parsing algorithm is compared with efficiencies of parsing algorithms based on other formalisms. Chapter 7 presents conclusions and future research.

Chapter 2

Related and Previous Work

There is a large area of research called Syntactic Pattern Recognition [16] that deals with the problem of using grammars to specify and recognize patterns. Picture Recognition is a particular application of it. Although this approach to picture recognition has much in common with specification and parsing of VPLs using grammars, it differs in the following ways:

- The purpose of Picture Recognition is to recognize pictures or images which are usually photographs of natural or manmade objects. Whereas VPLs are aimed at programming using graphical symbols which are predefined by the language.

- Picture Recognition just aims at recognizing input pictures nothing else whereas parsing process of VPLs usually carries out semantic actions associated with the selected production rules.

- Picture Recognition usually has two phases: Conversion of input image into a convenient representation and recognition of this pattern representation. VPL

parser on the other hand can take graphic function sequence corresponding to VPL sentence from Postscript file and parse it.

Following section presents a critic review of some of the grammatical formalisms used for Syntactic Pattern Recognition[1]. Section 2.2 presents a survey of previous work done in the area of specification and parsing of VPLs.

2.1 Syntactic Pattern Recognition

The following are different approaches to Syntactic Pattern Recognition, grouped according to input pattern representation.

- String Grammars: In this approach a picture is represented as a string of primitives. A primitive denote an elementary object, label or an operator. Examples of string based systems are Picture Description Language and MIRABELLE.
- Coordinate based systems: In this approach symbols with embedded coordinate values are used to represent pictures. Picture Processing Grammar falls in this category.
- Besides strings other higher order structures are used as basis for Syntactic Pattern Recognition. Tree Grammars, Plex Grammars, Web and Graph Grammars are based on this approach.
- Other approaches such as ESP[3] system are also used.

We will briefly discuss some of these formalisms.

[1]Some of these formalisms have also been used for VPLs

2.1.1 Picture Description Language

Shaw [31] used Picture Description Language (PDL for short) for describing and recognizing pictures. In a PDL system a picture is assumed to consist of several connected components (primitives). Each of these primitives belongs to a particular class. A primitive is assumed to have two connection points called its head and tail. For sake of convenience a primitive is represented by a directed arrow going from its tail to head. The arrow is labelled by the class name. A primitive can be connected to other primitives only at its connection points viz head and tail. Thus a picture can be represented as a directed graph where edges (arrows) denote the primitives and nodes denote the connections.

Primitive structural description of such a graph (or in other words the picture) is given by a linear string of primitive class names and connection operators. A connection operator could be one of the following:

- $+$, which connects head of one primitive to tail of another.
- -, which connects heads of two primitives.
- $\times$, which connects tails of two primitives.
- $*$, which connects both heads and tails of two primitives.
- $\neg$ unary operator, or / unary operator (unary operators are used to move head and tail of a picture).

The primitive structural description of a picture (or of its corresponding graph) is also called a PDL expression. To clarify these concepts let us consider an example[2].

[2]This example is reproduced from [31]

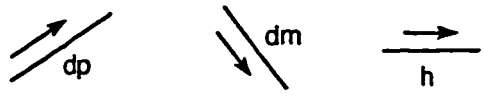

Figure 2.1: Primitive classes for line drawing of character A.

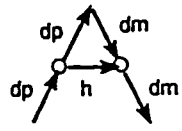

Figure 2.2: Directed graph representation of character A.

Example 2.1: Let us try to give the PDL expression for line drawing of character 'A'. Primitive classes for this example are shown in Figure 2.1. The directed graph representation of 'A' using these primitives is shown in Figure 2.2. PDL expression corresponding to the graph of Figure 2.2 is dp+(((dp+dm)*h)+dm).

In PDL system recognition of a picture proceeds as follows:
Picture is first segmented into primitive elements. Then connections between primitives are determined and PDL expression for it is created (this may involve two steps, construction of structural graph and derivation of PDL expression from it). Finally the PDL expression is parsed using a general parsing technique (eg., Earley's method). The underlying grammatical formalism used for describing valid PDL expressions is context free grammar. It is obtained by extending PDL expressions. This approach cannot be used in VPLs due to the following reasons.

- The concatenation of picture elements is the only relationship that can easily be expressed. In VPLs other goemetric relations among picture elements such as horizontal and vertical adjacency, overlapping, and enclosures are also desired.

- The process of transforming a picture into a string is a complex and time consuming one.

2.1.2 MIRABELLE

MIRABELLE [24] like PDL, is a picture description and recognition system based on context free grammar. Pictures are described by strings having primitive picture elements and relationship operators. In this case also, a primitive is assumed to have two connection points. In addition to coincidence operators MIRABELLE includes topological operators which describe goemetric relations between disjoint elements of a picture.

The parser for MIRABELLE uses a combination of top down and bottom up approaches. It starts somewhere inside the picture string and proceeds outwards, unlike other parsing algorithms which proceed left-to-right or right-to-left. MIRABELLE is an improvement over PDL in terms of expressiveness of grammars and sophistication of parsing algorithm. However MIRABELLE is not desirable as a grammar/parser mechanism for VPLs due to the following reasons:

- Parsing of pictures having disjoint elements may need backtracking.

- The grammar mechanism cannot be extended by language designer. Introduction of new operators requires modification to the parser.

2.1.3 Picture Processing Grammars

Chang [7] developed this formalism for describing 2D pictures. In fact it can be used as a basis for picture recognition, picture generation, and picture transformation.

A picture is viewed (represented) as a set of (symbol, associate vector) pairs, where associate vector denotes the spatial attributes of the symbol. Hierarchical structure of pictures is defined using a production set. Each production in this set consist of a rewrite rule and a partially computable function. The function maps associate vectors of r.h.s. elements to associate vectors of l.h.s. element. Chang classified these grammars as hierarchical and nonretracing.

Hierarchical grammars are those for which the production set can be partitioned into production blocks R_1, R_2, ...R_n such that if α appears as l.h.s. of a production in R_i then it should not appear on r.h.s. of a production in any of the R_js where j < i.

Nonretracing grammars are those for which if $\alpha \stackrel{*}{\Longrightarrow} U$ and $V \Longrightarrow U$ then $\alpha \stackrel{*}{\Longrightarrow} V$ for any picture U. This implies that nonretracing grammars may be parsed deterministically in a bottom up fashion.

A parsing algorithm is given in the reference for unambigious and nonretracing grammars. This algorithm is encorporated into a universal picture analyzer. The parsing analyzer works both in retracing and nonretracing modes. However in retracing mode it is not efficient.

The drawback of this formalism is that, it can only be applied to a limited set of languages and parsing based on it is not efficient. Complexity of parsing algorithm for a restricted form of picture processing grammar for 2D mathematical expressions is $O(N^2)$. This formalism is used is SIL [29].

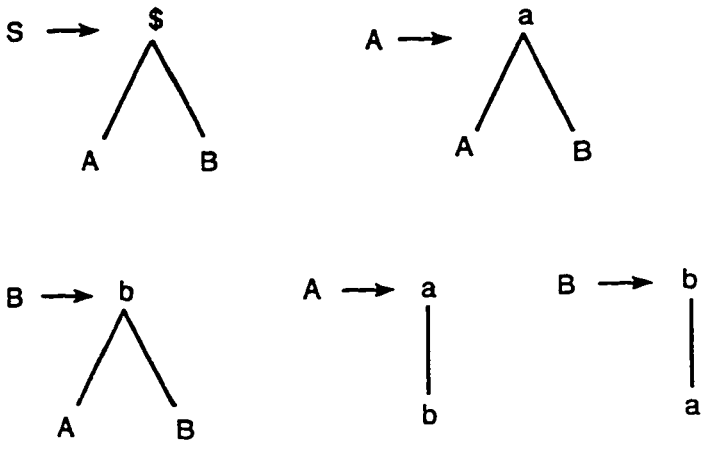

Figure 2.3: Production set of tree grammar of example 2.2.

2.1.4 Tree Grammars

Tree Grammars [15] are a natural extension of string grammars. Formally a *tree grammar* is defined as a 4-tuple (V, r, P, S) where (V, r) is a finite ranked alphabet such that $V_T \subseteq V$ and $V = V_T \cup V_N$. V_T and V_N are sets of terminals and nonterminals respectively. P is a finite set of productions of the form $\Phi \rightarrow \Psi$ where Φ, $\Psi \epsilon T_V$, and T_V denotes set of trees over alphabet (V, r). $S \subseteq T_V$ is a finite set of axioms. Rank of a symbol denoted by r specifies number of children of the node labelled by that particular symbol. Let us consider an example to clarify the definition[3].

Example 2.2: The tree grammar G_t = (V, r, P, S) where V = {S, a, b, $, A, B}, V_T = {$\stackrel{a}{\rightarrow}$, $\downarrow$b, \$}, r(a) = {2, 1, 0}, r(b) = {2, 1, 0}, r(\$) = 2, and P shown in Figure 2.3; generates pattern shown in Figure 2.4.

In tree grammar approach a picture is represented as a tree. The nodes of the

[3]This example is reproduced from [15]

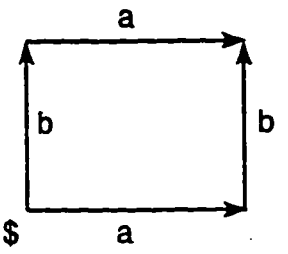

Figure 2.4: A pattern belonging to the language generated by the tree grammar of example 2.2.

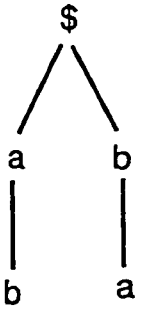

Figure 2.5: Tree representation of pattern in figure 2.4.

tree are labelled by symbols of V. Each symbol (or node) has an associated primitive. The edges of a tree denotes the connections between primitives. The tails of children primitives are assumed to be connected to the head of father primitive (like String Grammars here also a primitive is assumed to have two connection points head and tail). For example, the pattern in Figure 2.4 can be represented in tree form as shown in Figure 2.5

The disadvantage of tree grammars is that they still consider primitive as an object with two connection points. The primitives can only be connected at these connection points. Other compositions of primitives, such as enclosures and over-

lappings, which are desired in VPLs, cannot easily be expressed.

2.1.5 Plex Grammars

In String and Tree Grammars each primitive has two connection points whereas in Plex Grammars [14] it has arbitrary number 'n' of connection (attaching) points. A symbol of this type is termed as n attaching point entity (NAPE for short). The structure obtained by connecting NAPEs is called a plex structure. A plex structure is represented as three strings: an ordered list of components, a list of joints, and a list of tie-points. A connection of attaching points of two or more NAPEs is called a joint. The attaching points which are free are termed as tie-points.

Language of plex structures is specified using a plex grammar. A *plex grammar* is formally defined as a 6-tuple (V_N, V_T, P, S, Q, q_0) where

V_T is a finite nonempty set of NAPEs called the terminal vocabulary,

V_N is a finite nonempty set of NAPEs called the nonterminal vocabulary,

$V_T \cap V_N = \emptyset$,

P is a finite set of productions or replacement rules,

$S \epsilon V_N$ is a special NAPE called the initial NAPE,

Q is a finite set of identifiers,

$Q \cap (V_T \cup V_N) = \emptyset$,

$q_0 \epsilon Q$ is a special identifier called the null identifier.

The identifiers in Q are used to refer to the attaching points of the NAPEs. No two attaching points of the same NAPE can have the same identifier. The null identifier q_0 is used as a place marker and is not associated with any attaching points.

Chemical structures, electrical circuits, and a variety of other physical systems are described using Plex Grammars in the reference. However this formalism cannot be used for VPLs due to the following reasons:

- Plex Grammars are not suitable for describing pictures having disjoint or overlapped elements.
- They are difficult to understand and specify.
- Efficient parsers do not exist for this formalism.

2.1.6 Web and Graph Grammars

A web is a directed graph with labels on nodes. A *web grammar* [16] is a four tuple G = (N, T, S, P), where N and T are the nonterminal and terminal sets respectively, S is a set of initial webs, and P is a set of web productions. A web production is of the form $\alpha \rightarrow \beta$, E where α and β are webs (or subwebs), and E specifies how the nodes of β are to be connected to the neighbours of α. The following example, reproduced from [16], clarifies the definition.

Example 2.3: Consider a Web Grammar G = (N, T, S, P) where N = {A}, T = {a, b, c}, S = {A}, and P is as shown in Figure 2.6. The language of this grammar is the set of all webs of the form shown in Figure 2.7

A special form of web grammar having terminal set T of single element is called a *graph grammar*. All the nodes in a graph will have the same labels (unlabelled graph). Graph Grammars have been used by different researchers to represent pictures. Parsing techniques for Web and Graph Grammars are not efficient [19].

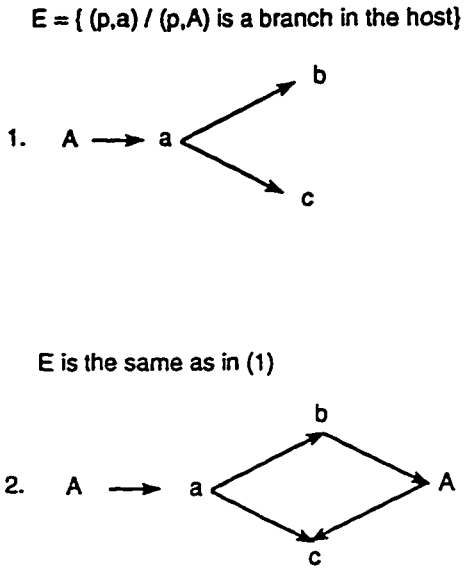

Figure 2.6: Web grammar productions of example 2.3.

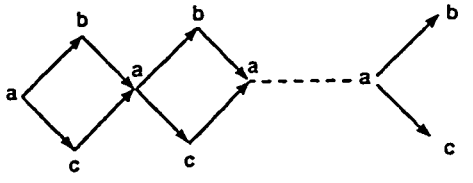

Figure 2.7: A sentence belonging to the language generated by web grammar of example 2.3.

2.1.7 ESP^3

The Extended Snobol Picture Pattern Processor (ESP^3) is a pattern recognition system designed for generating, recognizing, and manipulating two-dimensional line drawings [30]. It attempts to extend Snobol from strings and patterns of strings to pictures and pattern of pictures. It has a pattern matcher which tries to recognize a picture by scanning it in top-to-bottom and left-to-right order.

This approach cannot be applied to VPLs since it involves an exhaustive search of all possible pictures while matching against the input picture. Another complication is the need to create a new instance for every matched subpicture.

2.2 Previous Work on VPLs

This section is a review of some of the pioneering works on grammatical formalisms and parsing techniques for VPLs.

2.2.1 SILICON Compiler

Syntactic, Interactive, Learning, Icon-Oriented-System Compiler [29] (SILICON Compiler for short) is a software system for the specification, interpretation, prototyping and generation of Icon Oriented Systems. It is based on the concept of generalized icon. A generalized icon is represented as (X_m, X_i) where X_m is called the logical part (meaning), and X_i is called the physical part (image). SILICON system basically provides two functions: Icon Interpretation and Icon System Generation.

The Icon Interpreter takes a visual sentence as input, parses it according to the corresponding VPL grammar, and produces a visual concept. It then performs the task corresponding to the visual concept.

The Icon System Generator takes Icon System Definition as input and produces a realized Icon-Oriented System as output. Icon System Definition is basically the definition of visual language. It consists of G (Icon System), ID (Icon Dictionary), OD (Operator Dictionary), and ETAG (Extended Task Action Grammar). A brief description of each of these components is given below:

- G is a formal syntactic specification of the Icon System. It uses picture grammar to specify the physical part of the icons (concrete syntax of visual language). A picture grammar is a context free grammar where terminal symbols include both primitive picture elements and spatial operators. SILICON provides three spatial operators viz horizontal concatenation (&), vertical concatenation (^), and spatial overlay (+).

- ID is a database of elementary icons.

- OD has iconic operators, which are special icons to carry out some specific actions. These actions include

  - Constructing the image of a resultant icon.
  - Constructing conceptual graph of resultant icon.
  - Synthesizing descriptions.
  - Determining label of resultant icon.
  - Determining common attributes of icons and so on.

- ETAG provides a formal method for describing the user's conceptualization of intended tasks.

The Icon Interpreter parses an input iconic sentence in multiple steps. Initially the input picture is analyzed and converted into a string of terminal symbols (primitive picture elements and spatial operators). Example 2.4 will clarify this point.

Example 2.4: Consider the iconic sentence in Figure 2.8. It will be analyzed and converted to the string 'box+cross' by the Icon Interpreter. This string will then be parsed according to the given grammar.

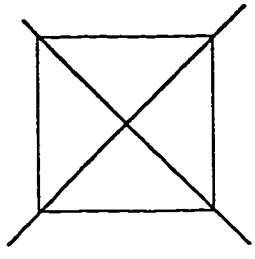

Figure 2.8: An iconic sentence.

SILICON compiler has the following drawbacks.

- When converting picture to string during parsing identical subpicture will be resolved differently in different contexts. This is not desirable in VPLs. An Example highlighting this drawback is given in [19].
- A complex visual sentence cannot be specified since only three operators are provided.
- The analysis phase (first phase of parser) is really complex and time consuming.

2.2.2 Visual Grammars

Lakin [22] advocated the use of Visual Grammars for describing Visual Programming Languages. A *visual grammar* is basically a context free grammar, annotated to indicate the spatial arrangement of picture elements. The right hand side of each production is a spatial template indicating the picture components and spatial arrangement among them. The r.h.s. elements can be visual literals (i.e., terminals) or nonterminals. Example 2.5 will clarify this point.

Example 2.5: Figure 2.9 shows the productions of a visual grammar for generating a family of bar charts. In the figure the connections between the line over the template and the elements of the right hand side describes the parse tree structure.

In this formalism an input sentence is viewed as spatial arrangement of text-graphic symbols where each text-graphic symbol is either a fragment of text or a primitive graphical element. A top down spatial parser based on this formalism is also presented. It works as follows:
Given a region of space and a desired element, the parser first marks the target region. Then it sorts the elements of the target region based on their upper left corners and stores them in a flat list. After that it chooses a production with the desired element as left hand side. An empty result tree template corresponding to r.h.s. of the chosen production is constructed. The parser then tries to fill the nodes of the tree template from the elements of sorted flat list. Terminal elements are moved from sorted list to tree template nodes. Nonterminals are expanded by repeating the above procedure. If a rule cannot be applied correctly ,i.e., if the

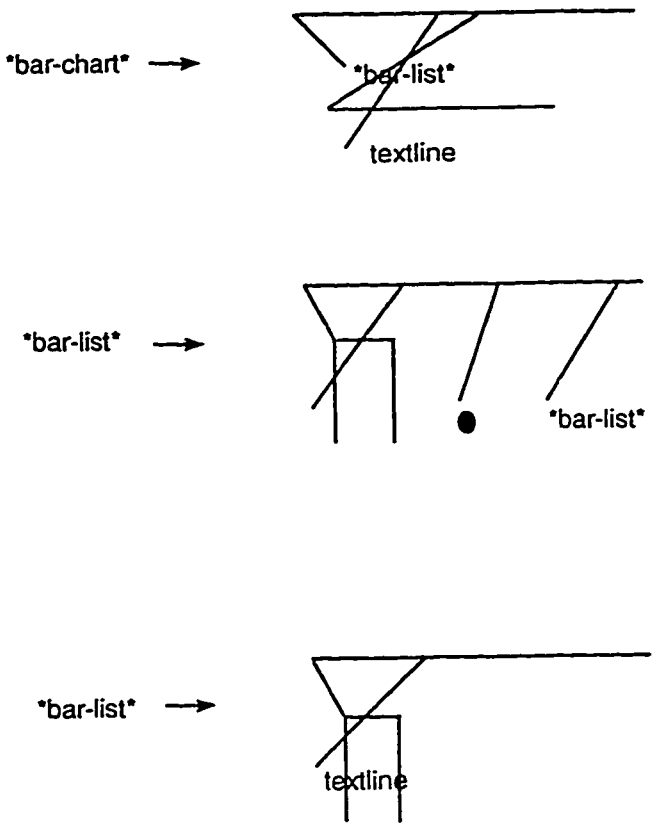

Figure 2.9: Visual grammar productions for a family of barcharts.

elements do not fit in the tree template, the parser backtracks and tries another rule (any alternate production, in case it exists for the desired element). This involves backtracking not only over the rules but also over the application of rules. It is not clear whether Lakin's parser is capable of doing this, even if it is, it could require exponential time [19]. Further the spatial relationships within a production are not formally specified. These must be precisely defined to correctly specify the syntax of VPLs.

2.2.3 Positional Grammars

Positional Grammars are used as grammatical formalism for specification and parsing of Visual Programming Languages in [9, 18]. For sake of convenience the definition of positional grammar is reproduced from [9] below. A *positional grammar* is a 6-tuple G = (N,T,S,P,POS,SP) where

- N is a finite set of nonterminals,
- T is a finite set of terminals,
- S is the starting symbol,
- POS is a set of positional operators (SP,p_1,p_2,.....p_n),
- SP is the starting position, and it is always the first element of the right hand side of the productions whose left hand side is the starting symbol S,
- P is a finite set of productions where each production has the form $A \rightarrow x_1 q_1 x_2 q_2 \ldots x_m$ where $A \epsilon N$, each x_i is in N $\cup$ T and each q_i is in POS.

This means that in a derivation step A is to be replaced by the symbols x_i, i=1,m; where x_i is placed at a position q_{i-1} relative to x_{i-1}.

Example 2.6[4] will clarify the definition.

Example 2.6: Let us give positional grammar for 2D arithmetic expressions with binary operators + and - This positional grammar can be written as G = (N, T, S, P, POS, SP) where

- N = {E, T, F},
- T = {+, -, (,), id},
- S = E,
- POS = {SP, HOR, VER}, where HOR and VER denote horizontal and vertical spatial relations,
- SP = random,
- P =
  - E := SP E HOR + HOR T | T,
  - T := T VER - VER F | F,
  - F := (HOR E HOR) | id.

The sentence $\frac{(id+id)}{id}+id$ is generated by the above grammar. A parsing algorithm called DR parser ('D' stands for driven scanning of the input and 'R' for rightmost

[4]This example is reproduced from [9]

derivation in reverse) is also given in [9]. This is a generalization of LR parser. As with LR parser the model of a DR parser is given by

- DR parsing program,
- DR parse table,
- Stack,
- Input, and
- Output.

The differences between an LR parser and a DR parser are mainly in the parse table, in the access to the input, and consequently in the parsing program. The parse table besides having action and goto fields has an additional field called position. This field specifies the position of next symbol to be accessed in a particular state (state on top of the stack). Working of the parser is similar to the LR parser except that it accesses the next symbol at the position indicated in position field of state on top of the stack. DR parse table generation algorithm is also given in [9]. It is similar to LR parse table generation method except for few modifications. The disadvantage of this parsing method is that it may not parse languages generated by ambigious positional grammars. In such cases the position field of a state may have more than one value[5].

An attempt to parse languages generated by ambigious Positional Grammars is done in [18]. The definition of positional grammar is slightly modified. The input picture is represented by an array of tokens, list of positions and token indices, and starting

[5]Example discussed in [9]

position. A parsing algorithm is also given. This algorithm is basically an extension of Earley's generalized parser[6]. As it is evident the disadvantage of this method is its lack of efficiency. The time complexity of this algorithm is not given in [18].

2.2.4 Picture Layout Grammars (PLGs)

A picture layout grammar [19] is basically a restricted form of attributed multiset grammar. Attributed multiset grammars are described in detail in [19]. In PLGs a visual sentence is represented as a multiset of attributed symbols. The attributes of a symbol denote its location, size, color and so on. Example 2.7 shows a iconic sentence and its representation as multiset of attributed symbols.

Example 2.7: The visual sentence in Figure 2.10 can be represented as {square[1,2,4,5], square[8,2,11,5], arrow[4,3,8,3]}

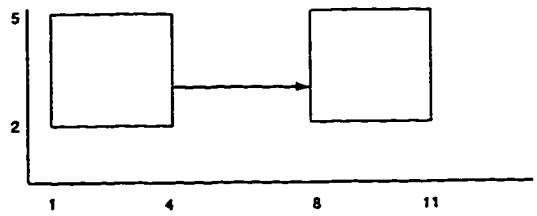

Figure 2.10: A sample visual sentence.

Formal definition of picture layout grammar reproduced from [19] is as follows. A *picture layout grammar* is a non-overlapping, finitely representable attributed multiset grammar G = (N,Σ, s, I, D, P) where

[6]Detailed algorithm with an example is given in [18]

- N is a finite set of nonterminal symbols,
- Σ is a finite set of terminal symbols,
- s ϵ N is a start symbol,
- I is a finite set of attribute identifiers called the parsing attributes,
- D is a set of attribute domains,
- P is a set of productions of the form (R, SF, C) where R is a rewrite rule, SF is a set of semantic functions, C is a set of constraints.

And the following are true

1. Every symbol x ϵ N $\cup$ Σ has (atleast) the following four distinguished parsing attributes - lx, by, rx, ty; where lx and by denote the lower left corner of the symbol or of rectangle enclosing it, and rx and ty denote, the upper right corner of the symbol (or of rectangle enclosing it),
2. If p = (R, SF , C) ϵ P, with R = A $\rightarrow$ X/Y, then for all α ϵ {lx, rx, by, ty }, the semantic function SF contains an assignment A.α = x.β where x ϵ X $\cup$ Y and β ϵ {lx,rx,by,ty}. In other words, p copies attributes lx, rx, by, ty of left hand side from similar (but not necessarily corresponding) attributes of elements of the right hand side.

A grammar G is said to be nonoverlapping if for all multisets M ϵ A(G), rank(M) $\leq$ 1. In other words no two symbols in any multiset M ϵ A(G) should be exactly the same (i.e., same symbol and attributes). A(G) refers to the set of all multisets which can be analyzed (recognized) using G.

A grammar G is said to be finitely representable if for all attributes a ϵ I, cardinality(a) with respect to A(G) is finite. In other words each attribute a ϵ I should take on finite number of values in all M ϵ A(G).

Although PLGs are a special form of attributed multiset grammars they have slightly different syntax for productions. A production in PLG consists of three parts: the rewrite rule, the semantic function, and the constraints. The rewrite rule has two possible forms:

1. N $\rightarrow$ X.

2. N $\rightarrow$ OP(X_1,.....X_m).

where N is a nonterminal and X is a terminal or nonterminal. In the first form the nonterminal N on l.h.s. is replaced by X which is a terminal or nonterminal of r.h.s. (possibly some constraints may be enforced or some attributes may be modified). In the second form N on l.h.s. is replaced by a composition of symbols X_1,X_2,.....X_m. These symbols may be terminals or nonterminals, OP represents the composition operator. A table of built in composition operators is given in [19]. The second part of production is the semantic function. The semantic function computes the attributes of aggregate object on l.h.s. from the attributes of constituent objects on r.h.s. Each built in composition operator has semantic functions associated with it. These built in functions compute the 4 attributes lx, by, rx, ty. Apart from this, user can include additional semantic functions in a production. When a production is applied its built in semantic functions are evaluated first followed by the evaluation of user specified semantic functions.

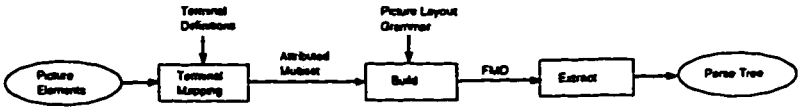

Figure 2.11: Phases of a PLG based parser.

Constraints form the third part of productions. These specify the conditions to be met for application of a production. Each built in composition operator has associated constraints. In addition to these, user can specify his own constraints. Constraints are usually relational or boolean expressions over attributes of symbols. All the constraints are ANDed before application of a production.

The author has also given a parsing method for PLGs. This parsing method has several phases as shown in Figure 2.11. Initially the picture (visual sentence) is converted to a attributed multiset. The build phase of the parser takes this representation of input sentence and picture layout grammar of VPL, and builds a Factored Multiple Derivation (FMD) structure. The FMD is a type of factored parse tree and represents the set of all possible analyses of the program. The FMD structure may contain invalid analyses because of the bottom up approach of the build phase. The second phase (extract phase) takes FMD and extracts a single analysis from it. Detailed algorithms for these phases are given in [19].

The disadvantage of this method is that it is very inefficient. The parser was applied to actual language (Statecharts) and the performance was found to be bounded by $O(N^3)$, where N is number of symbols in the input sentence.

2.2.5 Relational Grammars

The family of grammatical formalisms, which represent visual sentences as multiset of icons with certain relationships over them, are loosely termed as Relational Grammars [33]. This family includes grammatical formalisms such as Relation Grammars (to be discussed in chapter 3) and Picture Layout Grammars. The languages generated by Relational Grammars are termed as Relational Languages. A sentence in a relational language is represented as a multiset of symbols together with a set of relations over them (over the symbols in multiset). A bottom up parsing algorithm for a relatively unrestricted class of Relational Grammars is given in [21]. This algorithm has the advantage that input objects (icons of visual sentence) can be scanned and composed in any order. The disadvantage of the algorithm is the lack of efficiency. To describe an efficient parsing algorithm a subclass of Relational Grammars called Fringe Relational Grammars are proposed in [33].
Fringe Relational Grammars (FRGs for short) describe visual sentences as partially ordered multiset (md-poset for short) of symbols. Each relation in the relationship set R_e, imposes a different partial ordering on the symbols of multiset. A partially ordered multiset is formally defined in [33] as follows:

Md-poset: A multidimensional partially-ordered multiset (md-poset for short) is a multidimensional multiset (Ψ,R_1,R_2,...,R_n) such that each R_1,R_2,...,R_n is a strict partial order on the set Ψ. The set Ψ is the multiset of symbols or icons.

An md-poset is actually a variant (or can be considered as special form) of multidimensional multiset (md-set for short). An md-set is formally defined in [33] as

below:

Md-set: A multidimensional multiset (md-set) is an n-tuple (Ψ,R_1,R_2,...,R_n) such that R_1,R_2,...,R_n are binary relations on the multiset Ψ.

Sometimes an md-set A = (Ψ,R_1,R_2,...,R_n) is written as the set of tuples {(r_1 x_a x_b), (r_1 x_c x_d),...,(r_n x_e x_f)} where each r_i is a symbol indicating the relation in position i of R_1, R_2,...,R_n and each $x_\alpha \in \Psi$. The minimal and maximal elements of an md-poset, arranged according to a particular partial order, imposed by a particular relation r ϵ R_e, are termed as fringe elements of md-poset in relation r. Each relation in the set R_e imposes a different ordering on the symbols of md-poset. With respect to each relation in R_e, we have a different set of fringe elements, which are minimal and maximal elements of md-poset ordered according to that relation. These elements are written as 'relation_name - fringe_name ', where relation_name is the name of a relation r ϵ R_e, and fringe_name can be min or max (denoting minimal or maximal element respectively). Let us consider the following examples to clarify the notion of fringe elements.

Example 2.8: Let us assume that 'left-of' relation holds between two successive elements of a string. Let v='abcd' be a string. Then the fringe elements of v, ordered according to the 'left-of' relation are as follows:

left-of min = a

left-of max = d

Example 2.9: Let us assume that 'left-of' and 'above' are spatial relations of a fringe relational grammar for 2D patterns. Let

w =

```
      e
    a b d
      c
```

be a 2D pattern. The fringe elements of w, for 'left-of' and 'above' relations are as follows:

left-of min = a

left-of max = d

above min = e

above max = c

In FRG all the visual sentences and intermediate sentential forms of a derivation are md-posets. Since all the sentential forms of FRG are md-posets, the r.h.s. of each production should also be defined as an md-poset. Further the fringe elements of left hand side symbol of production should be expressed in terms of the fringe elements of r.h.s. symbols. As a result an FRG production is of the form $A \rightarrow \alpha/\beta/F$ where A is a nonterminal; α is a string of symbols (terminals and/or nonterminals); β is a set of constraints specifying partial orderings on symbols of α ; F is a set of statements which assigns Fringe values to l.h.s. symbol, for each relation $r \in R_e$. The following is the formal definition of FRG reproduced from [33].

Fringe Relational Grammars: Fringe Relational Grammars (FRGs) are a 5-

tuple G = (N,Σ,S,R_e,P) where

1. N is a finite set nonterminal symbols,

2. Σ is a finite set of terminal symbols, disjoint from N,

3. S is a distinguished symbol in N called the start symbol,

4. R_e is a finite set of relation symbols called the expander relation symbols,

5. P is a finite set of productions of the form $A \rightarrow \alpha/\beta/F$, where

   $A \epsilon N$;

   $\alpha \epsilon (n \mid \sigma)^+$ where $n \epsilon N$, $\sigma \epsilon \Sigma$;

   β is a set of constraints of the form (r_e i j) where $r_e \epsilon R_e$ and i,j are members of α referenced by position and β imposes a strict partial order on elements in string α;

   F is a function from $R_e \times$ {min,max} to symbols in α such that F(x, min) is a member of the minimal elements in α ordered by relation x and F(x, max) is a member of the maximal elements in α ordered by relation x.

Let us consider the following example, reproduced from [33] to clarify the definition.

Example 2.10: The fringe relational grammar (FRG) which describes a ternary tree of symbols, where the mother is centered and above each row of horizontally aligned daughters, is written as FRG1 = (N, Σ, S, R_e, P) where

N = {S, Row, Subtree},

Σ = {id},

S = S,

R_e = {left-of, above},

P = {P1: S → Subtree

P2: Subtree → id Row

(above 1 2)

(left-of_min 0) = 1

(left-of_max 0) = 1,

P3: Row → Subtree Subtree Subtree

(left-of 1 2)

(left-of 2 3)

(above-min 0) = 2

(above-max 0) = 2,

P4: Subtree → id}.

An expression in L(FRG1) is shown in Figure 2.12.

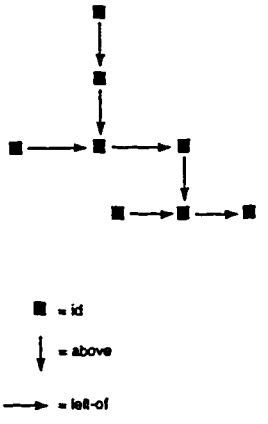

Figure 2.12: An expression in L(FRG1).

Let us look at a production of FRG1 to clarify what it specifies. Let us consider production 2.

P2: Subtree $\rightarrow$ id Row

(above 1 2)

(left-of min 0) = 1

(left-of max 0) = 1

This production says that in a derivation of FRG1, nonterminal 'Subtree' will be expanded to 'id' and 'Row'. The symbols of production are referenced by indices in β and F part of the production. 'Subtree' is referenced as 0, 'id' is referenced as 1, and 'Row' as 2. The β part of the production i.e. (above 1 2) says that id is to be placed above the nonterminal Row, in a derivation. The F part, i.e.,

(left-of min 0) = 1,

(left-of max 0) = 1,

states the following: The first part of F part, i.e., (letf-of min 0) = 1, states that the minimal fringe element of the terminal 'id', in relation left-of, is to be passed onto(assigned as) minimal fringe element of 'Subtree', in relation left-of. Similarly the second part substitutes the max fringe elements.

An attempt to specify an Earley style predictive parsing algorithm for FRGs is done in [33]. The parsing algorithm specified in [33] is incomplete. Some of the subroutines of the algorithm are not specified. So this algorithm cannot be considered as a parsing or even a recognition algorithm. Another drawback of FRGs is that the relationship set R_e has only binary relations. Due to this the expressivenss of FRGs is somewhat restricted.

Chapter 3

Relation Grammars

3.1 Relation Grammars

Relation Grammars(RGs for short) are introduced in [6] as extensions of traditional Textual Grammars. They are reproduced again in [13, 17, 23]. In this formalism, information about replacement mechanism of productions and derivation process is made explicit. Production rules not only specify the set of symbols replacing a particular nonterminal, but also the spatial relationships between those symbols. For example [13] the following textual language production.

p: $A \rightarrow x_1x_2...x_n$

can be written as an RG production as follows.

$A \rightarrow \{x_1,x_2,...,x_n\}\{left(x_1,x_2),left(x_2,x_3),...,left(x_{n-1},x_n)\}$

The relation 'left' is used to express linear concatenation of symbols. The RG production says that in a sentential form, a particular occurence of nonterminal A will be replaced by the symbols $\{x_1,x_2,...,x_n\}$, provided the constraints $\{left(x_1,x_2),left(x_2,x_3),$ $....,left(x_{n-1},x_n)\}$ are satisfied. It also says that when this production is applied, the

generated symbols $\{x_1,x_2,...,x_n\}$ should be arranged according to the constraints $\{left(x_1,x_2),left(x_2,x_3),...,left(x_{n-1},x_n)\}$.

Another set of rules called evaluation rules are also provided which specify how the generated symbols $\{x_1,x_2,...,x_n\}$ relate to the neighbours of A. Assuming 'y' to be the symbol on the left and 'z' to be the symbol on the right of A, in a sentential form, evaluation rules can be written as

$left(y,A)$:- $(A \Rightarrow \{x_1,x_2,...,x_n\}),left(y,x_1)$

$left(A,z)$:- $(A \Rightarrow \{x_1,x_2,...,x_n\}),left(x_n,z)$

This formalism will be of little use for textual languages. To appreciate its usefulness let us consider an example.

Example 3.1:[1]. Consider the following 2D pattern which denotes an intermediate visual language sentential form.

x_1 x_2 A

x_3

Assuming usual interpretation of 'left' and 'above' relations, the sentential form can be written as $\{x_1,x_2,A,x_3\}\{left(x_1,x_2),left(x_2,A),above(A,x_3)\}$

If we have a production $A \rightarrow \{y_1,y_2\}$ then the exact form of sentence obtained after applying this production is not known. This is because we do not have information about spatial arrangement of symbols y_1 and y_2. Instead if we have an RG produc-

[1]This example is reproduced from [13]

tion

$A \rightarrow \{y_1,y_2\}$ $\{above(y_2,y_1)\}$

Then atleast we have some idea of the resulting 2D pattern. It is clear to us that in the resulting sentential form, y_2 is to be placed above y_1. But this information is not sufficient. With just this information the following two sentences are possible.

$$\begin{array}{c} y_2 \\ x_1\ x_2\ y_1 \\ x_3 \end{array} \qquad\qquad \begin{array}{c} x_1\ x_2\ y_2 \\ y_1 \\ x_3 \end{array}$$

Therefore we need a way of specifying how the generated symbols(symbols on r.h.s. of a production) are to be related to the neighbours of the replaced symbol. This is done by the use of evaluation rules. An evaluation rule has a constraint in its left hand side, and a set of constraints plus a structural condition in the r.h.s. The l.h.s. constraint specifies the relationship which holds between the replaced symbol and its neighbours. The r.h.s. constraints specify how the l.h.s. relationship is going to change when the production specified by the structural condition is applied. In the preceeding example, if we have the evaluation rules.

$left(x_2,A)$:- $(\{A \Rightarrow \{y_1,y_2\}),left(x_2,y_1)$

$above(A,x_3)$:- $(\{A \Rightarrow \{y_1,y_2\}),above(y_1,x_3)$

then we get the following sentence (after application of A-production and evaluation rules)

$$\begin{array}{c} y_2 \\ x_1\ x_2\ y_1 \\ x_3 \end{array}$$

If we have the evaluation rules

left(x_2,A) :- ({A $\Rightarrow$ {y_1,y_2}),left(x_2,y_2)

above(A,x_3) :- ({A $\Rightarrow$ {y_1,y_2}),above(y_1,x_3)

then we get the following sentence

x_1 x_2 y_2

y_1

x_3

By now we have an idea how the productions and evaluation rules of RG are used to describe multidimensional patterns. Before reproducing the definition of RG grammar, let us see how a sentence is represented in RG grammar.

RG sentence: Given[2] an alphabet of terminal symbols V_T and a set of relation symbols V_R, a sentence w over V_T and V_R is a pair (T,RT) such that T is a multiset of symbols in V_T and RT is a set of relation instances in V_R over symbols in T.

Example 3.2: For example, the sentence

w = $\diamond^1\diamond^2$

$\diamond^3$

can be represented as w = (T, RT) over V_T = {$\diamond$} and V_R = {left, up}, where T = {$\diamond^1$, $\diamond^2$, $\diamond^3$} and RT = {left($\diamond^1$,$\diamond^2$), up($\diamond^2$,$\diamond^3$)}.

[2]This definition is reproduced from [6]

The following is the definition of relation grammar reproduced from [6].

Relation Grammar: Relation Grammar(RG) is defined as a 6-tuple G = (V_N, V_T, V_R, S, P, R) where V_N is a set of nonterminal symbols, V_T is a set of terminal symbols(icons), V_R is a set of relation symbols, $S \epsilon V_N$ is the starting symbol, P is a finite set of productions. Each production has the form A ::= M,R_1, where $A \epsilon V_N$, M is a multiset of symbols in $V_N \cup V_T$, and R_1 is a set of constraints over the symbols in M which must hold for the production to be applied. R is a set of evaluation rules, to validate constraints.

To clarify the definition of relation grammar let us consider an example.

Example 3.3: Consider a grammar which generates a language consisting of sequences of symbols '◇' in descending stair step pattern. A sample sentence of this language is w, shown earlier. Such a language can be described by the relation grammar G = (V_N, V_T, V_R, S, P, R) where V_N = {S}, V_T = {◇}, V_R = {left, up, adjacent}, P contains the productions:

1. S ::= {◇, S} {adjacent(◇, S)}
2. S ::= {◇}

and R contains the following rules:

1. adjacent(◇, S) :- left(◇, S)
2. adjacent(◇, S) :- up(◇, S)

3. left($\diamond^1$, S^1) :- ($S^1 \Rightarrow \{\diamond^2\}$) $\vee$ ($S^1 \Rightarrow \{\diamond^2, S^2\}$), left($\diamond^1$,$\diamond^2$)

4. up($\diamond^1$, S^1) :- ($S^1 \Rightarrow \{\diamond^2\}$) $\vee$ ($S^1 \Rightarrow \{\diamond^2, S^2\}$), up($\diamond^1$,$\diamond^2$)

The productions characterize sequences of adjacent symbols $\diamond$. First production states that S on input symbol $\diamond$ is to be expanded to $\{\diamond,S\}$, provided the constraint adjacent($\diamond$,S) is satisfied. Second production expands S to(or replaces S by) $\diamond$(there is no constraint in this case). Evaluation rules are used to validate the constraints appearing on r.h.s. of productions(if any). These rules are of two types:

1. Constraints over certain symbols are written in terms of other constraints (relationships) over the same symbols. Like for example evaluation rules 1 and 2 just rewrites constraint adjacent($\diamond$, S) as left($\diamond$, S) or up($\diamond$, S). This means that two symbols are considered adjacent if the first one appears to the left or above the second one.

2. The constraints over nonterminals are written in terms of (reduced to) constraints over terminals (or constraints over terminals and nonterminals), provided some conditions are satisfied. These conditions specify the expansion of a nonterminal. For example rule 3 says that the constraint left($\diamond^1$, S^1) will be reduced to left($\diamond^1$, $\diamond^2$), if S^1 is replaced by $\diamond^2$ or $\{\diamond^2, S^2\}$, in a derivation step.

Let us demonstrate the use of evaluation rules using the visual sentence w. The parse tree obtained for w is depicted in Figure 3.1. Each node in the parse tree specifies application of a production A := M, R_1, and is labelled by constraints in R_1 which must be satisfied in order to apply the production. At level 0 we apply

the production S = {◇, S} {adjacent(◇, S)}. The constraint at this node i.e. adjacent(◇, S) is validated by applying the evaluation rules as follows:

adjacent($◇^1$, S^2) $\rightarrow_1$ left($◇^1$, S^2) $\rightarrow_3$ left($◇^1$,$◇^2$).

The expression r $\rightarrow_k$ r^1 denotes that the relation instance r is reduced to r^1 via the k-th evaluation rule. Since the last relation instance, left($◇^1$,$◇^2$) belongs to w, the original constraint is valid. The relation instance adjacent($◇^2$, S^3) at level 1 is validated in a similar way, and so on.

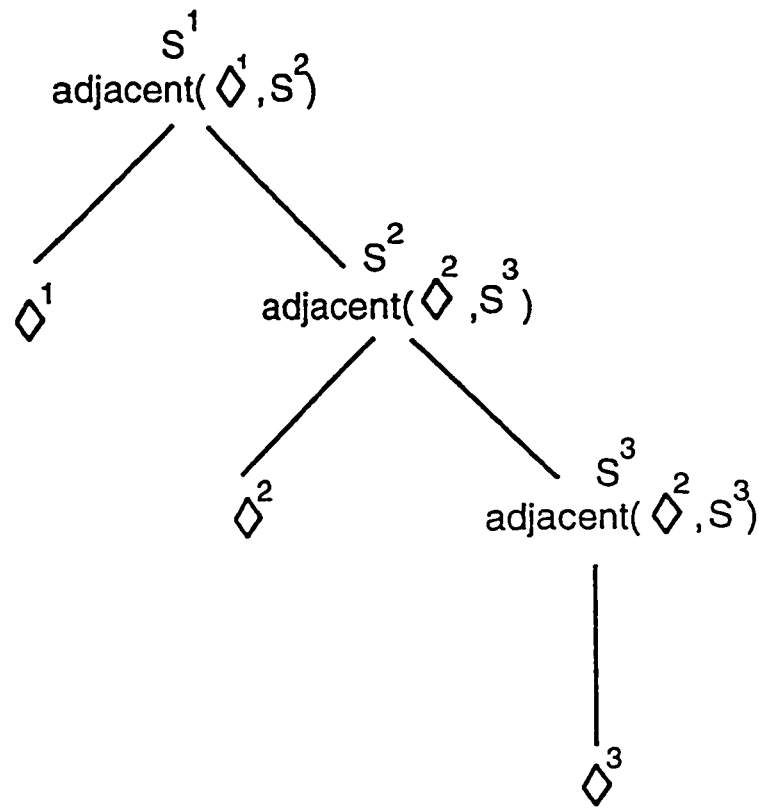

Figure 3.1: Parse tree for sentence w.

A parsing algorithm based on RG formalism is given in [13, 17]. The parsing

algorithm works bottom up and in two phases. In the first phase the input sentence is checked to see whether symbols in T belongs to V_T or not, and relation instances defined on them (i.e.the set RT) are valid or not. A relation instance is said to be valid if the relation it uses belongs to the set V_R and the relation is defined over symbols in T. In the second phase it is verified whether the input sentence can be derived by means of production rules whose constraints are valid. An outline of the parsing algorithm (basically the second phase) is reproduced below from [17] .

Method: The parsing algorithm takes as input an RG G = (V_N,V_T,V_R,S,P,R) and an input sentence u = (sym,rel) = ($\{a_1,a_2,...,a_m\}$,$\{r_1(s_1,t_1),...,r_n(s_n,t_n)\}$) where $r_j \in V_R$, $a_i \in V_T$ and s_j, $t_j \in$ sym, for j = 1,2,...,n and i = 1,2,...,m. It produces parse tree for u if it is grammatically correct, else prints error message. The parsing algorithm presented here applies to RG with binary relations[3] but it can be easily generalized to n-ary relations as shown in example 3.3 (for 3-ary relations). Initially the input sentence u is stored in sym-acc and vin-acc respectively (sym is stored in sym-acc and rel is stored in vin-acc). At a particular step in parsing, a set of nodes(symbols) are selected from sym-acc and reduced. Constraints in vin-acc are updated accordingly. The procedure stops when a complete parse tree is constructed in sym-acc and vin-acc is empty. The algorithm is formalised below.

1. Let sym-acc be initially set to $\{\theta_1,\theta_2,...,\theta_m\}$, where each θ_i is a tree with a single node labeled by a_i, for i = 1,...,m, and vin-acc = rel;

[3]A binary relation grammar is an RG where each r ϵ V_R is a binary relation

2. If sym-acc contains only one tree θ whose root is labeled by S and vin-acc = $\emptyset$, then output θ;

3. Nondeterministically choose a subset $\pi = \{\beta_1,\beta_2,...,\beta_k\}$ of sym-acc and a production A::= $\{c_1,c_2,...,c_k\},\rho_A$ such that, if π-root = $\{b_1,b_2,...,b_k\}$ is the set of the roots of the trees in π and π-vin is the set of constraints in vin-acc whose arguements are all in π-root, then

   (a) $\{b_1,b_2,...,b_k\} = ren_p(\{c_1,c_2,...,c_k\})$, i.e., π-root is a renaming of the set $\{c_1,c_2,...,c_k\})$, and

   (b) $ren_p(\rho_A) \subseteq \pi$-vin, i.e., the corresponding renaming of ρ_A is included in π-vin. If no such production exists, then output "error";

4. Nondeterministically choose a set of evaluation rules $e_1,e_2,...,e_q$ with

   $e_1 : s_1(X_1,Y_1)$:- $(A \Rightarrow \{c_1,c_2,...,c_k\}),s_{1,1}(x_{1,1},y_{1,1}),...,s_{1,k1}(x_{1,k1},y_{1,k1})$

   .

   .

   .

   $e_q : s_q(X_q,Y_q)$:- $(A \Rightarrow \{c_1,c_2,...,c_k\})$, $s_{q,1}(x_{q,1},y_{q,1}),...,s_{q,kq}(x_{q,kq},y_{q,kq})$ where $s_{i,g}(x_{i,g},y_{i,g}) \in$ vin-acc and $X_i = A$ and $x_{i,g},y_{i,g} \in \{c_1,c_2,...,c_k\} \cup \{Y_i\}$ or $Y_i = A$ and $x_{i,g},y_{i,g} \in \{c_1,c_2,...,c_k\} \cup \{X_i\}$ for i = 1,2,...,q and g = 1,2,...,k_i;

5. Delete $\beta_1,\beta_2,...,\beta_k$ from sym-acc and add the tree with root A and subtrees

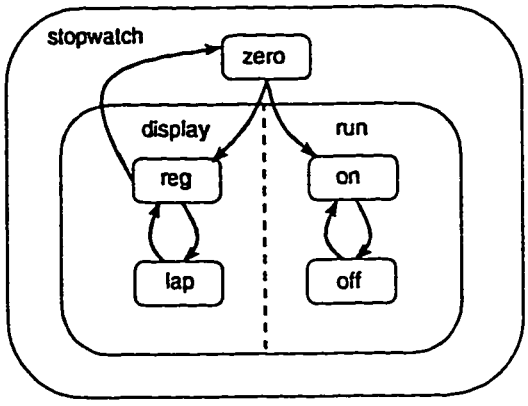

Figure 3.2: States and Transitions in a statechart.

$\beta_1,\beta_2,...,\beta_k$. Delete all the elements in ρ_A and nondeterministically delete the constraints $s_{i,g}(x_{i,g},y_{i,g})$ from vin-acc and add to vin-acc the constraints $s_1(X_1,Y_1),...,s_q(X_q,Y_q)$;

6. Go to step 2.

Let us consider an example RG and apply parsing algorithm to it. Let us give RG formalism for the types of statecharts shown in Figure 3.2 [17].

Example 3.4: RG formalism for the types of statecharts shown in Figure 3.2 can be written as G = (V_N,V_T,V_R,STCH,GrP,R) where V_N = {STCH,SUPST,ARR}, V_T = {text,□,→,|}, V_R = {Connect,Cont,Left,Start,End}, the set GrP of productions is

1. STCH ::= {$SUPST^1$,$STCH^1$,ARR^1}{Connect(ARR^1,$SUPST^1$,$STCH^1$)}

2. STCH ::= {$SUPST^1$}

3. SUPST ::= $\{\square^1,text^1\}\{Cont(\square^1,text^1)\}$

4. SUPST ::= $\{\square^1,text^1,STCH^1\}\{Cont(\square^1,text^1),Cont(\square^1,STCH^1)\}$

5. SUPST ::= $\{\square^1,text^1,text^2,|^1,STCH^1,STCH^2\}\{Cont(\square^1,text^1),Cont(\square^1,text^2),$
   $Cont(\square^1,|^1),Cont(\square^1,STCH^1),Cont(\square^1,STCH^2),Left(STCH^1,|^1),Left(|^1,STCH^2)\}$

6. ARR ::= $\{\rightarrow^1\}$

7. ARR ::= $\{\rightarrow^1,ARR^1\}$

and R contains following evaluation rules

1. $Connect(ARR^1,SUPST^1,STCH^1)$:- $(ARR^1 \Rightarrow \{\rightarrow^1,ARR^2\})$,
   $Connect(\rightarrow^1,SUPST^1,STCH^1)$, $Connect(ARR^2,SUPST^1,STCH^1)$

2. $Connect(ARR^1,SUPST^1,STCH^1)$:- $(ARR^1 \Rightarrow \{\rightarrow^1\})$,
   $Connect(\rightarrow^1,SUPST^1,STCH^1)$

3. $Connect(\rightarrow^1,STCH^1,STCH^2)$:- $(STCH^1 \Rightarrow \{ARR^1,SUPST^1,STCH^3\})$,
   $Connect(\rightarrow^1,STCH^1,STCH^2)$

4. $Connect(\rightarrow^1,STCH^1,STCH^2)$:- $(STCH^2 \Rightarrow \{ARR^1,SUPST^1,STCH^3\})$,
   $Connect(\rightarrow^1,SUPST^1,STCH^1)$

5. $Connect(\rightarrow^1,SUPST^1,STCH^1)$:- $(SUPST^1 \Rightarrow \{\square^1,text^1\})$, $Connect(\rightarrow^1,\square^1,STCH^1)$

6. $Connect(\rightarrow^1,SUPST^1,STCH^1)$:- $(SUPST^1 \Rightarrow \{\square^1,text^1,text^2,|^1,STCH^2,STCH^3\})$,
   $Connect(\rightarrow^1,STCH^2,STCH^1)$, $Connect(\rightarrow^1,STCH^3,STCH^1)$

7. $Connect(\rightarrow^1,SUPST^1,STCH^1)$:- $(SUPST^1 \Rightarrow \{\square^1,text^1,text^2,|^1,STCH^2,STCH^3\})$,
   $Connect(\rightarrow^1,STCH^2,STCH^1)$

8. Connect($\rightarrow^1$,$\square^1$,STCH^1) :- (STCH^1 $\Rightarrow$ {SUPST^1}), Connect($\rightarrow^1$,$\square^1$,SUPST^1)

9. Connect($\rightarrow^1$,$\square^1$,SUPST^1) :- (SUPST^1 $\Rightarrow$ {$\square^2$,text^1}), Start($\rightarrow^1$,$\square^1$), End($\rightarrow^1$,$\square^2$)

10. Connect($\rightarrow^1$,$\square^1$,SUPST^1) :- (SUPST^1 $\Rightarrow$ {$\square^2$,text^1}), Start($\rightarrow^1$,$\square^2$), End($\rightarrow^1$,$\square^1$)

11. Cont($\square^1$,STCH^1) :- (STCH^1 $\Rightarrow$ {SUPST^1,STCH^2,ARR^1}), Cont($\square^1$,SUPST^1), Cont($\square^1$,STCH^2), Cont($\square^1$,ARR^1)

12. Cont($\square^1$,SUPST^1) :- (SUPST^1 $\Rightarrow$ {$\square^2$,text^1}), Cont($\square^1$,$\square^2$)

13. Cont($\square^1$,SUPST^1) :- (SUPST^1 $\Rightarrow$ {$\square^2$,text^1,STCH^1}), Cont($\square^1$,$\square^2$)

14. Cont($\square^1$,SUPST^1) :- ($\square^2$,text^1,|^1,STCH^1,STCH^2}), Cont($\square^1$,$\square^2$)

15. Left(STCH^1,|^1) :- (STCH^1,STCH^2,ARR^1}), Left(SUPST^1,|^1), Left(STCH^2,|^1)

Here only a subset of R is given. The remaining evaluation rules can be constructed similarly. For example, rules for the constraint Cont($\square^1$,STCH^1) with the structural condition (STCH^1 $\Rightarrow$ {SUPST^1}), and for Cont($\square^1$,ARR^1), are analagous to rules 11-14. Rules for Left(STCH^1,|^1), Left(|^1,STCH^1) etc., are similar to rule 15. Let us apply the parsing algorithm to statechart in Figure 3.2. The statechart in Figure 3.2 is represented in RG as a sentence w shown below:

w = {$\rightarrow^1$,$\rightarrow^2$,text^1,$\square^1$,$\square^2$,text^2,|^1,text^3,$\rightarrow^3$,$\rightarrow^4$,$\square^3$,text^4,$\square^4$,text^5,$\rightarrow^5$,$\rightarrow^6$,$\square^5$,text^6,$\square^6$,
text^7,$\square^7$,text^8}{Cont($\square^1$,text^1),Cont($\square^1$,$\square^2$),Cont($\square^1$,$\square^7$),Cont($\square^7$,text^8),
Cont($\square^2$,text^2),Cont($\square^2$,text^3),Cont($\square^2$,|^1),Cont($\square^2$,$\rightarrow^3$),Cont($\square^2$,$\rightarrow^4$),Cont($\square^2$,$\square^3$),
Cont($\square^2$,$\square^4$),Cont($\square^2$,$\rightarrow^5$),Cont($\square^2$,$\rightarrow^6$),Cont($\square^2$,$\square^5$),Cont($\square^2$,$\square^6$),Start($\rightarrow^1$,$\square^7$),
End($\rightarrow^1$,$\square^3$),End($\rightarrow^1$,$\square^5$),Start($\rightarrow^2$,$\square^3$),End($\rightarrow^2$,$\square^7$),Cont($\square^3$,text^4),Cont($\square^4$,text^5),

Cont($\square^5$,text^6),Cont($\square^6$,text^7),Start($\rightarrow^3$,$\square^3$),End($\rightarrow^3$,$\square^4$),Start($\rightarrow^4$,$\square^4$),End($\rightarrow^4$,$\square^3$),
Start($\rightarrow^5$,$\square^5$),End($\rightarrow^5$,$\square^6$),Start($\rightarrow^6$,$\square^6$),End($\rightarrow^6$,$\square^5$),Left($\square^3$,$|^1$),Left($\square^4$,$|^1$),
Left($|^1$,$\square^5$),Left($|^1$,$\square^6$)}.

The first choice of the parsing algorithm that leads to a correct parse tree is shown in the following steps.

1. Initialization(Step 1):
   sym-acc = {$\rightarrow^1$,$\rightarrow^2$,text^1,$\square^1$,$\square^2$,text^2,$|^1$,text^3,$\rightarrow^3$,$\rightarrow^4$,$\square^3$,text^4,$\square^4$,text^5,$\rightarrow^5$,$\rightarrow^6$,$\square^5$,
   text^6,$\square^6$,text^7,$\square^7$,text^8}
   and vin-acc = {Cont($\square^1$,text^1),Cont($\square^1$,$\square^2$),Cont($\square^1$,$\square^7$),Cont($\square^7$,text^8),
   Cont($\square^2$,text^2),Cont($\square^2$,text^3),Cont($\square^2$,$|^1$),Cont($\square^2$,$\rightarrow^3$),Cont($\square^2$,$\rightarrow^4$),Cont($\square^2$,$\square^3$),
   Cont($\square^2$,$\square^4$),Cont($\square^2$,$\rightarrow^5$),Cont($\square^2$,$\rightarrow^6$),Cont($\square^2$,$\square^5$),Cont($\square^2$,$\square^6$),Start($\rightarrow^1$,$\square^7$),
   End($\rightarrow^1$,$\square^3$),End($\rightarrow^1$,$\square^5$),Start($\rightarrow^2$,$\square^3$),End($\rightarrow^2$,$\square^7$),Cont($\square^3$,text^4),Cont($\square^4$,text^5),
   Cont($\square^5$,text^6),Cont($\square^6$,text^7),Start($\rightarrow^3$,$\square^3$),End($\rightarrow^3$,$\square^4$),Start($\rightarrow^4$,$\square^4$),End($\rightarrow^4$,$\square^3$),
   Start($\rightarrow^5$,$\square^5$),End($\rightarrow^5$,$\square^6$),Start($\rightarrow^6$,$\square^6$),End($\rightarrow^6$,$\square^5$),Left($\square^3$,$|^1$),
   Left($\square^4$,$|^1$),Left($|^1$,$\square^5$),Left($|^1$,$\square^6$)}

2. The termination test(step 2) fails and the next step is executed.

3. The subset π = {$\square^7$,text^8} of sym-acc and production 3 are chosen(step 3):
   then π-root = {$\square^7$,text^8}, and π-vin = {Cont($\square^7$,text^8)};

4. Evaluation rule 9 is chosen to reduce the constraints Start($\rightarrow^2$,$\square^3$),End($\rightarrow^2$,$\square^7$)
   ϵ vin-acc, and evaluation rule 10 is chosen to reduce the constraints Start($\rightarrow^1$,$\square^7$),
   End($\rightarrow^1$,$\square^3$), and Start($\rightarrow^1$,$\square^7$),End($\rightarrow^1$,$\square^5$) (step 4);

5. The corresponding reduction step is applied(step 5) and the following sets are obtained: sym-acc = {$\rightarrow^1$,$\rightarrow^2$,text^1,$\square^1$,$\square^2$,text^2,$|^1$,text^3,$\rightarrow^3$,$\rightarrow^4$,$\square^3$,text^4,$\square^4$, text^5,$\rightarrow^5$,$\rightarrow^6$,$\square^5$,text^6,$\square^6$,text^7,SUPST^1}
and vin-acc = {Cont($\square^1$,text^1),Cont($\square^1$,$\square^2$),Cont($\square^1$,$\square^7$),Cont($\square^1$,SUPST^1), Cont($\square^2$,text^2),Cont($\square^2$,text^3),Cont($\square^2$,$|^1$),Cont($\square^2$,$\rightarrow^3$),Cont($\square^2$,$\rightarrow^4$),Cont($\square^2$,$\square^3$), Cont($\square^2$,$\square^4$),Cont($\square^2$,$\rightarrow^5$),Cont($\square^2$,$\rightarrow^6$),Cont($\square^2$,$\square^5$),Cont($\square^2$,$\square^6$), Connect($\rightarrow^1$,SUPST^1,$\square^3$),Connect($\rightarrow^1$,$\square^5$,SUPST^1),Connect($\rightarrow^2$,$\square^3$,SUPST^1), Cont($\square^3$,text^4),Cont($\square^4$,text^5),Cont($\square^5$,text^6),Cont($\square^6$,text^7),Start($\rightarrow^3$,$\square^3$), End($\rightarrow^3$,$\square^4$),Start($\rightarrow^4$,$\square^4$),End($\rightarrow^4$,$\square^3$),Start($\rightarrow^5$,$\square^5$),End($\rightarrow^5$,$\square^6$), Start($\rightarrow^6$,$\square^6$),End($\rightarrow^6$,$\square^5$),Left($\square^3$,$|^1$),Left($\square^4$,$|^1$),Left($|^1$,$\square^5$),Left($|^1$,$\square^6$)}

6. Go to step 2.

Further reduction steps are performed until sym-acc = {STCH} and vin-acc = {}. At this stage the test of step 2 succeeds and the tree is output as the complete parse tree.

It is evident that the parsing algorithm for RG is purely bottom up approach which tries all possible parses of an input sentence. Hence this algorithm is inefficient. Efficiency of parsing algorithm can be improved by using some sort of prediction which can rule out constituents that are well formed locally but have no chance of contributing to the complete parse. Prediction cannot be applied to RG because of nondeterminism in the grammar. Nondeterminism in RG is because of the following reasons: [6].

1. Possibility of alternative expansions for nonterminals.

2. Possible applicability of alternative evaluation rules when validating a constraint.

In order to describe an efficient parsing algorithm a subclass of RG called RG/1 is proposed [13, 17].

3.2 Relation Grammar/1

Relation Grammar/1 grammars (RG/1 grammars for short) are defined by imposing restrictions both on the form of productions and on the evaluation rules. Though RG/1 formalism is not as expressive as RG, it is capable of describing visual languages of practical use. The definition of RG/1 is reproduced from [17] below.

RG/1 grammar: An RG grammar G = (V_N, V_T, V_R, S, P, R) is said to be an RG/1 grammar, if and only if, the following occur.

1. An ordering is defined over the relation symbols in V_R.

2. Each production in G has the form

   $A ::= \{Y_1, Y_2, ...Y_k\}\ \{r_1(Y_1, Y_2), r_2(Y_1, Y_3), ..., r_{k-1}(Y_1, Y_k), ..., R_1, R_2, ...R_h\}$,

   where $Y_1 \in V_T$, $Y_i \in V_N \cup V_T$, i = 2,3,...,k; $r_j \in V_R$, j = 1,....k-1; and $r_i \neq r_j$, whenever $i \neq j$, $i,j \in \{1,....k\text{-}1\}$,

   i.e., there exists one symbol, say Y_1, which is bounded to any other symbol Y_j by means of a relation r_{j-1}; $R_1, R_2, ...R_h (h \geq 0)$ denote possible additional constraints over the symbols of the set $\{Y_2, ...Y_k\}$. A production has a special terminal symbol Y_1 on its right-hand side, the primary symbol, which

is related to the other symbols through different relations; the other symbols might in turn be linked by arbitrary relations.

3. Two productions having the same symbol on the left hand side cannot have the same primary symbol.

4. Given a nonterminal symbol A ϵ V_N, if t(X_1,A),....,t(X_n,A) are constraints over A in G (with X_1,....,X_n $\epsilon V_N \cup V_T$ and t ϵV_R), then the set of evaluation rules for t(X_1,A),....,t(X_n,A) are constructed in such a way that the context is preserved where the replacement of A takes place. It means that when applying a production to A, the constraint t(X_j,A) is replaced by a constraint involving X_j and one of the produced symbols, for j = 1,...,n. Thus for each production

   A ::= {Y_1,Y_2,...,Y_k} {r_1(Y_1,Y_2), r_2(Y_1,Y_3),...,r_{k-1}(Y_1,Y_k),...,R_1,R_2,...,R_h},

   the set of evaluation rules of G must contain the rules:

   t(X_1,A) :- (A $\Rightarrow$ {Y_1,Y_2,...,Y_k}),q_1(X_1,Z_1),

   .

   .

   .

   t(X_n,A) :- (A $\Rightarrow$ {Y_1,Y_2,...,Y_k}),q_n(X_n,Z_n),

   where Z_j = Y_i for some i ϵ {1,...,k},j = 1,...,n or when k = 1, the rules:

   t(X_1,A) :- (A $\Rightarrow$ {Y}),t(X_1,Y),

   .

   .

   .

   t(X_n,A) :- (A $\Rightarrow$ {Y}),t(X_n,Y).

To clarify the definition, the following example is reproduced from [13].

Example 3.5: Consider a language consisting of 2D grids of two kinds of trees. Trees of type ♠ are placed on the south-east frontier, remaining grid positions are filled by ♣ trees. A sample sentence of this language is v shown below:

v = ♣ ♣ ♠

♠ ♠ ♠

RG/1 for this language can be written as G = (V_N, V_T, V_R, S, P, R) = ({S, A, B}, {♣, ♠}, {x, y, z}, S, P, R). The x and y relations indicate the horizontal and vertical adjacency, repectively. The relation z denote a diagonal adjacency in NE-SW direction. The order on these relations is x<y<z. P consists of the following productions

S := {♣, B, A} {x(♣, B), y(♣, A), z(B,A)}

B := {♣, B, A} {x(♣, B), y(♣, A), z(B,A)}

B := {♠, A} {y(♠, A)}

A := {♣, A} {y(♣, A)}

A := {♠}

and R consists of

$x(♣^1, B^1)$:- $(B^1 \Rightarrow \{♣^2, B^2, A\})$, $x(♣^1, ♣^2)$

x(♣, B) :- $(B \Rightarrow \{♠, A\})$, x(♣, ♠)

$y(♣^1, A^1)$:- $(A^1 \Rightarrow \{♣^2, A^2\})$, $y(♣^1, ♣^2)$

y(♣, A) :- $(A \Rightarrow \{♠\})$, y(♣, ♠)

$z(B^1, A^1)$:- $(B^1 \Rightarrow \{♣^2, B^2, A^2\})$, $x(A^1, A^2)$

$z(B, A^1)$:- $(B \Rightarrow \{\spadesuit, A^2\})$, $x(A^1, A^2)$

$x(A^1, A^2)$:- $(A^1 \Rightarrow \{\clubsuit, A^3\})$, $x(\clubsuit, A^2)$, $z(A^2, A^3)$

$x(A^1, A^2)$:- $(A^1 \Rightarrow \{\spadesuit\})$, $x(\spadesuit, A^2)$

$x(\clubsuit^1, A^1)$:- $(A^1 \Rightarrow \{\clubsuit^2\}, A^2)$, $x(\clubsuit^1, \clubsuit^2)$

$x(\clubsuit, A)$:- $(A \Rightarrow \{\spadesuit\})$, $x(\clubsuit, \spadesuit)$

$z(A^1, A^2)$:- $(A^1 \Rightarrow \{\clubsuit, A^3\})$, $x(A^2, A^3)$

$z(A^1, A^2)$:- $(A^1 \Rightarrow \{\spadesuit\})$, $x(A^2, \spadesuit)$

$x(\spadesuit, A^1)$:- $(A^1 \Rightarrow \{\clubsuit, x(A^2\})$, $x(\spadesuit, \clubsuit)$

$x(\spadesuit^1, A)$:- $(A \Rightarrow \{\spadesuit^2\})$, $x(\spadesuit^1, \spadesuit^2)$

$y(\spadesuit, A^1)$:- $(A^1 \Rightarrow \{\clubsuit, A^2\})$, $y(\spadesuit, \clubsuit)$

$y(\spadesuit^1, A)$:- $(A \Rightarrow \{\clubsuit^2\})$, $y(\spadesuit^1, \spadesuit^2)$

$x(A^1, \spadesuit)$:- $(A^1 \Rightarrow \{\clubsuit, A^2\})$, $z(\spadesuit, \clubsuit)$

$x(A, \spadesuit^1)$:- $(A \Rightarrow \{\spadesuit^2\})$, $z(\spadesuit^1, \spadesuit^2)$.

In RG/1 grammars the introduction of primary symbol on r.h.s. of productions removes the nondeterminism from the production set P. At a particular parsing step, on a particular input symbol, only one production applies. The form of evaluation rules imposed in condition 4, guarantees that at each step it is possible to decide which evaluation rule to apply. This removes nondeterminism from the evaluation rules set R.

A top down parsing algorithm based on RG/1 is proposed. It takes as input an RG/1 grammar and a visual sentence represented as (T,RT). It produces a top down parse of the sentence if it is grammatically correct, else prints error message and stops. The algorithm is reproduced later on in this section. Before giving algorithm let

us give some definitions. Some of the definitions are reproduced from [13] in which RG/1 is introduced. Due to some editing errors in [13], most of the definitions are reproduced from [17], in which RG/1 are reproduced.

The following definition is reproduced from [13].

Sentential form: Given a relation grammar G = (V_N,V_T,V_R,S,P,R), the pair (M,R) where M is a list of terminal and nonterminal symbols and R is a list of relation instances, is a *sentential form* of G if there exist a derivation
$\{S\} \Rightarrow (M_1,R_1) \Rightarrow \ldots \Rightarrow (M_n,R_n) = (M,R)$.

The following definition is reproduced from [17].

Ordered Sentential form: Given a binary RG grammar G = (V_N,V_T,V_R,S,P,R), a sentential form (sym,rel) is said to be an *ordered sentential form* if and only if the following conditions hold

1. V_R is an ordered set, that is a total ordering $<$ is defined over its relation symbols;

2. there exists a unique element a ϵ sym, called the *least element*, such that there is no r ϵ V_R and no b ϵ sym such that r(b,a) ϵ rel;

3. For each b ϵ sym, if $r_1(b,c_1)$ ϵ rel and $r_2(b,c_2)$ ϵ rel for some c_1,c_2 ϵ sym, then $r_1 \neq r_2$.

Given an ordered sentential form (sym,rel), it is possible to recursively number the elements of sym as follows:

Let the least element of sym be numbered by 1. For i = 2,3,...,|sym| let 'a' be the element of sym numbered by i and let $r_1(a,a_1),r_2(a,a_2),...,r_k(a,a_k) \epsilon$ rel with $r_1 < r_2 < .... < r_k$. Then, number the elements $a_1,a_2,...,a_k$ that have not been numbered yet in a progressive way.

The following definition is reproduced from [17].

Regular left hand side derivation: A *regular left-hand side derivation* of an RG grammar $G = (V_N,V_T,V_R,S,P,R)$ is a derivation:

$(\{S\},\emptyset) \Rightarrow (sym_1,rel_1) \Rightarrow ... \Rightarrow (sym_n,rel_n)$.

in which (sym_i,rel_i) is an ordered sentential form, and at each step, a production is applied to the first nonterminal symbol in sym_i, for i = 1,...,n.

Only the above kind of derivation will be allowed in RG/1 grammars. This will become clear when we consider the parsing algorithm. The following two definitions are reproduced from [17].

Context identical symbols: Let (sym_1,rel_1) and (sym_2,rel_2) be sentential forms of an RG/1 grammar $G = (V_N,V_T,V_R,S,P,R)$, where $sym_1 = \{X_1,...,X_n\}$ and $sym_2 = \{Y_1,...,Y_n\}$. Two symbols $X_i \epsilon sym_1$ and $Y_i \epsilon sym_2$ are *context-identical* if the following conditions apply:

1. $X_i = Y_i$

2. If $s_1(X_i,Z_1),...,s_m(X_i,Z_m)$ and $t_1(Y_i,W_1),...,t_m(Y_i,W_m)$
   are the constraints involving X_i in rel_1 and Y_i in rel_2 respectively, then $s_j = t_j$ and $W_j = Z_j$, j = 1,...,m.

Potentially context identical symbols: Let (sym_1,rel_1) and (sym_2,rel_2) be sentential forms of an RG/1 grammar $G = (V_N,V_T,V_R,S,P,R)$, where $sym_1 = \{X_1,...,X_n\}$ and $sym_2 = \{Y_1,...,Y_n\}$. Two symbols $X_k \epsilon sym_1$ and $Y_k \epsilon sym_2$ *are potentially context identical*, if the following conditions apply:

1. $X_k = Y_k$

2. If $s_1(X_k,Z_1),...,s_m(X_k,Z_m)$ and $t_1(Y_k,W_1),...,t_m(Y_k,W_m)$ are the constraints involving X_k in rel_1 and Y_k in rel_2 respectively, then:
   if $Z_j \epsilon V_T$, then $s_j = t_j$, j = 1,...,m; if $W_j \epsilon V_N$ and $s_j \neq t_j$, then for each W_j a production $W_j ::= \rho,\sigma$ exists such that $Z_j \epsilon \rho$ and $s_j(X_k,Z_j)$ is in ρ.

Let q be the position of W_j in sym_2, then the quadruple (q,k,Z_j,s_j) is called *description of a potentially contextual identity (dpci).*

The parsing algorithm is reproduced from [13] in Figure 3.3. Input to the algorithm is an RG/1 grammar $G = (V_N,V_T,V_R,S,P,R)$ and a sentence (M,R), where $M = \{X_1,X_2,...,X_m\}$. Output is top down parse of the sentence (M,R) if it is grammatically correct, else error message.

Let $(M_0,R_0) = (\{S\},\emptyset)$ and $D = \emptyset$
for i = 1 to m do
begin
let Y_i be the i-th symbol in M_{i-1}
if $Y_i \in V_N$ then
{ find the production having Y_i, on the left hand
side and X_i, as primary symbol
if no production is applicable
then output *error* and stop
apply the selected production obtaining
sentential form (M_i,R_i) }
Let X and Y be the symbols occuring at the i-th
position in M and M_i, respectively
If X and Y are not context-identical and X and Y are potentially context-identical
then each description of potentially contextual identity
is added to the set D
Let Y ::= (M,R) be the selected production.
If there is a dpci (i,k,z_j,r_j) in D
then if $Z_j \in M$, $r_j(Y_k,z_j) \in R$ andY_k is the k-th
symbol in M,
then remove (q,k,z_j,r_j) from D
else output *error* and stop.

Figure 3.3: Parsing algorithm based on RG/1.

The parsing algorithm is applied to example 3.5 and steps are traced. Before that example 3.5 is reproduced again for the sake of convenience.

Example 3.5 (reproduced): Consider a language consisting of 2D grids of two kinds of trees. Trees of type ♠ are placed on the south-east frontier, remaining grid positions are filled by ♣ trees. A sample sentence of this language is v shown below:

v = ♣ ♣ ♠

♠ ♠ ♠

RG/1 for this language can be written as G = (V_N, V_T, V_R, S, P, R) = ({S, A, B}, {♣, ♠}, {x, y, z}, S, P, R). The x and y relations indicate the horizontal and vertical adjacency repectively. The relation z denote a diagonal adjacency in NE-SW direction. The order on these relations is x<y<z. P consists of the following productions

S := {♣, B, A} {x(♣, B), y(♣, A), z(B,A)}

B := {♣, B, A} {x(♣, B), y(♣, A), z(B,A)}

B := {♠, A} {y(♠, A)}

A := {♣, A} {y(♣, A)}

A := {♠}

and R consists of

$x(♣^1, B^1)$:- $(B^1 \Rightarrow \{♣^2, B^2, A\})$, $x(♣^1, ♣^2)$

x(♣, B) :- $(B \Rightarrow \{♠, A\})$, x(♣, ♠)

$y(♣^1, A^1)$:- $(A^1 \Rightarrow \{♣^2, A^2\})$, $y(♣^1, ♣^2)$

y(♣, A) :- $(A \Rightarrow \{♠\})$, y(♣, ♠)

$z(B^1, A^1)$:- $(B^1 \Rightarrow \{♣^2, B^2, A^2\})$, $x(A^1, A^2)$

$z(B, A^1)$:- $(B \Rightarrow \{♠, A^2\})$, $x(A^1, A^2)$

$x(A^1, A^2)$:- $(A^1 \Rightarrow \{♣, A^3\})$, $x(♣, A^2)$, $z(A^2, A^3)$

$x(A^1, A^2)$:- $(A^1 \Rightarrow \{♠\})$, $x(♠, A^2)$

$x(♣^1, A^1)$:- $(A^1 \Rightarrow \{♣^2\}, A^2)$, $x(♣^1, ♣^2)$

x(♣, A) :- $(A \Rightarrow \{♠\})$, x(♣, ♠)

$z(A^1, A^2)$:- $(A^1 \Rightarrow \{♣, A^3\})$, $x(A^2, A^3)$

$z(A^1, A^2)$:- $(A^1 \Rightarrow \{♠\})$, $x(A^2, ♠)$

$x(\spadesuit, A^1)$:- $(A^1 \Rightarrow \{\clubsuit, x(A^2\})$, $x(\spadesuit, \clubsuit)$

$x(\spadesuit^1, A)$:- $(A \Rightarrow \{\spadesuit^2\})$, $x(\spadesuit^1, \spadesuit^2)$

$y(\spadesuit, A^1)$:- $(A^1 \Rightarrow \{\clubsuit, A^2\})$, $y(\spadesuit, \clubsuit)$

$y(\spadesuit^1, A)$:- $(A \Rightarrow \{\clubsuit^2\})$, $y(\spadesuit^1, \spadesuit^2)$

$x(A^1, \spadesuit)$:- $(A^1 \Rightarrow \{\clubsuit, A^2\})$, $z(\spadesuit, \clubsuit)$

$x(A, \spadesuit^1)$:- $(A \Rightarrow \{\spadesuit^2\})$, $z(\spadesuit^1, \spadesuit^2)$

Steps involved in parsing v are as follows.

Initially: By considering the ordering of relations, v is organised as (M,R) =
$(\{\clubsuit^1,\clubsuit^2,\spadesuit^2,\spadesuit^1,\spadesuit^3,\spadesuit^4\}, \{x(\clubsuit^1,\clubsuit^2),y(\clubsuit^1,\spadesuit^2),x(\spadesuit^2,\spadesuit^3),x(\clubsuit^2,\spadesuit^1),y(\clubsuit^2,\spadesuit^3),$
$x(\spadesuit^3,\spadesuit^4),y(\spadesuit^1,\spadesuit^4)\})$; $(M_0,R_0) = (\{S\},\emptyset)$; $D = \emptyset$;

Iteration 1: $Y_1 = S$, $X_1 = \clubsuit^1$; production 1 is applied and we get
$(M_1,R_1) = \{\clubsuit^1,B^1,A^1\}\{x(\clubsuit^1,B^1),y(\clubsuit^1,A^1),z(B^1,A^1)\}$
$D = \{(2,1,\clubsuit^2,x),(3,1,\spadesuit^2,y)\}$

Iteration 2: $Y_2 = B_1$, $X_2 = \clubsuit^2$; production 2 is applied
$(M_2,R_2) = \{\clubsuit^1,\clubsuit^2,A^1,B^2,A^2\}\{x(\clubsuit^1,\clubsuit^2),y(\clubsuit^1,A^1),x(A^1,A^2),x(\clubsuit^2,B^2),y(\clubsuit^2,A^2),z(B^2,A^2)\}$
$D = \{(3,1,\spadesuit^2,y),(4,2,\spadesuit^1,x),(5,2,\spadesuit^3,y)\}$

Iteration 3: $Y_3 = A^1$, $X_3 = \spadesuit^2$; production 5 is applied
$(M_3,R_3) = \{\clubsuit^1,\clubsuit^2,\spadesuit^2,B^2,A^2\}\{x(\clubsuit^1,\clubsuit^2),y(\clubsuit^1,\spadesuit^2),x(\spadesuit^2,A^2),x(\clubsuit^2,B^2),y(\clubsuit^2,A^2),z(B^2,A^2)\}$
$D = \{(4,2,\spadesuit^1,x),(5,2,\spadesuit^3,y)(5,3,\spadesuit^3,x)\}$

Iteration 4: $Y_4 = B^2$, $X_4 = \spadesuit^1$; production 3 is applied

$(M_4,R_4) = \{\clubsuit^1,\clubsuit^2,\spadesuit^2,\spadesuit^1,A^2,A^3\}\{x(\clubsuit^1,\clubsuit^2),y(\clubsuit^1,\spadesuit^2),x(\spadesuit^2,A^2),x(\clubsuit^2,\spadesuit^1),$
$y(\clubsuit^2,A^2),x(A^2,A^3),y(\spadesuit^1,A^3)\}$

$D = \{(5,2,\spadesuit^3,y)(5,3,\spadesuit^3,x),(6,4,\spadesuit^4,y)\}$

Iteration 5: $Y_5 = A^2$, $X_5 = \spadesuit^3$; production 5 is applied

$(M_5,R_5) = \{\clubsuit^1,\clubsuit^2,\spadesuit^2,\spadesuit^1,\spadesuit^3,A^3\}\{x(\clubsuit^1,\clubsuit^2),y(\clubsuit^1,\spadesuit^2),x(\spadesuit^2,\spadesuit^3),x(\clubsuit^2,\spadesuit^1),$
$y(\clubsuit^2,\spadesuit^3),x(\spadesuit^3,A^3),y(\spadesuit^1,A^3)\}$

$D = \{(6,4,\spadesuit^4,y),(6,5,\spadesuit^4,x)\}$

Iteration 6: $Y_6 = A^3$, $X_6 = \spadesuit^4$; production 5 is applied

$(M_6,R_6) = \{\clubsuit^1,\clubsuit^2,\spadesuit^2,\spadesuit^1,\spadesuit^3,\spadesuit^4\}\{x(\clubsuit^1,\clubsuit^2),y(\clubsuit^1,\spadesuit^2),x(\spadesuit^2,\spadesuit^3),x(\clubsuit^2,\spadesuit^1),$
$y(\clubsuit^2,\spadesuit^3),x(\spadesuit^3,\spadesuit^4),y(\spadesuit^1,\spadesuit^4)\}$; $D = \{\}$

The time complexity of the above algorithm as derived in [13] is O(n log n), where n is the number of elementary icons in a visual sentence. The drawback of this parsing algorithm is that it accepts visual sentences with total ordering on elementary icons (terminal symbols), as input. This total ordering is to be performed based on the ordering of spatial relations. For example v was ordered as $\{\clubsuit^1,\clubsuit^2,\spadesuit^2,\spadesuit^1,\spadesuit^3,\spadesuit^4\}\{x(\clubsuit^1,\clubsuit^2),y(\clubsuit^1,\spadesuit^2),x(\spadesuit^2,\spadesuit^3),x(\clubsuit^2,\spadesuit^1),y(\clubsuit^2,\spadesuit^3),x(\spadesuit^3,\spadesuit^4),$ $y(\spadesuit^1,\spadesuit^4)\}$. If the input sentences are not totally ordered the parsing algorithm fails to recognize them. This ordering has to be performed during lexical analysis phase. The need to order the sentences, according to the ordering of spatial relations, com-

plicates the lexical analysis phase. The lexical analyzer has to provide tokens in a particular order. Another drawback is that each sentential form in the derivation should be totally ordered. This complicates the parsing procedure.

In [17], in addition to parsing algorithms for RG and RG/1, the expressiveness of RG grammars is compared with graph grammar formalisms. It is shown that RG formalism is atleast as powerful as two well known graph grammar formalisms viz Context Free Graph Grammar (CFGG for short) and Arc-labelled directed node-label controlled graph grammar (edNLC graph grammar for short). Methods for transforming CFGGs and edNLCs to RGs are also given[4].

3.3 1NS-RG

In [23] another restricted form of RG called "One Non-terminal S-item Relation Grammar (1NS-RG for short)" is introduced. In this formalism any constraint whether it appears on r.h.s. of production or evaluation rule has atmost one non-terminal. Formal definition of 1NS-RG reproduced from [23] is as follows.

1NS-RG: A grammar G = (V_N,V_T,V_R,S,P,R) is said to be a 1NS-RG if for every s-production N ::= σ,ρ ϵ P and for every r-production r ::= [p,j]ρ ϵ R, ρ does not contain r-items involving two nonterminals s-items, i.e., if t(v^i,w^j) ϵ ρ, then v ϵ V_T or w ϵ V_T.

In the above definition s-production means a production belonging to the set P, and r-production means an evaluation rule belonging to the set R. s-item means a

[4]Examples are given in [17]

terminal or nonterminal, r-item means a constraint (instance of a relation). The following example of 1NS-RG is reproduced from [23].

Example 3.6: Let us give a 1NS-RG description of a window system. It can be written as G = ({S,W,B},{ww,cw,tb},{ins,sca,ins_sca,right},S,P,R) where ww, cw, tb stand for working window, confirm window, and toolbar, respectively. P contains the following s-productions:

[1] $S ::= \{tb^1,W^2,S^3\}\{ins(W^2,tb^1),right(tb^1,S^3)\}$

[2] $S ::= \{cw^1,W^2\}\{right(W^2,cw^1),sca(cw^1,W^2)\}$

[3] $S ::= \{tb^1,cw^1,W^3\}\{ins\_sca(W^3,tb^1),right(tb^1,cw^2),sca(cw^2,tb^1)\}$

[4] $W ::= \{ww^1\}$

and R contains the r-productions:

$ins(W,tb) ::= [4]\{ins(ww^1,tb)\}$

$ins\_sca(W,tb) ::= [4]\{ins(ww^1,tb),sca(tb,ww^1)\}$

$right(W,cw) ::= [4]\{right(ww^1,cw)\}$

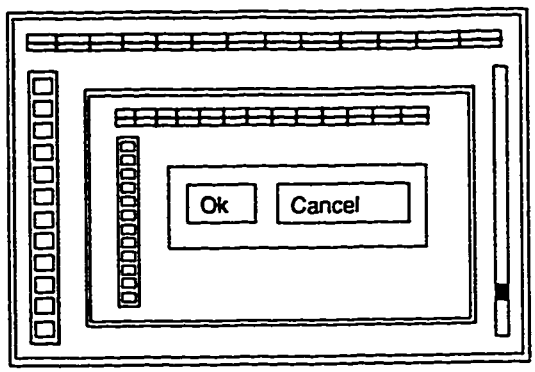

Figure 3.4: A sample generated by 1NS-RG of example 3.6.

sca(cw,W) ::= [4]{sca(cw,ww^1)}

right(tb,S) ::= [1]{right(tb,W^2)}

right(tb,S) ::= [2]{right(tb,W^2)}

right(tb,S) ::= [3]{right(tb,W^3)}

right(tb,W) ::= [3]{right(tb,ww^1)}

A sample sentence w of L(G) is shown in Figure 3.4.

1NS-RG satisfies context freeness property. This means that the order of applying derivation steps in a derivation of a 1NS-RG does not change the resulting sentential form. This property is used to design a predictive parsing algorithm. This algorithm is essentially a recognizer of visual sentences. It does not generate parse

tree for the input sentence. It takes a visual sentence represented as a pair (T,RT) and checks whether it can be accepted or not. By accepted it means whether it belongs to L(G), for a particular 1NS-RG G or not. The algorithm is an extension of Earley's algorithm [10]. One remarkable feature of this algorithm is that it accepts visual sentences without any total or partial ordering on their symbols. It constructs all the top down parses of a sentence w simultaneously. The parsing process consists of three basic operations predicting, scanning, and completing. The order of applying these operations here is predicting, scanning, and completing which is different from Earley's method, where the order is predicting, completing, and scanning. Also the way these operations work is slightly different from the Earley's method. When a symbol in input sentence w is scanned a set of states X_i is constructed. Each state $s = <p,Rs,j,f,w_s>$ in the set contains

1. An s-production of the form [p] = [A ::= σ,ρ].

2. A set Rs of r-productions of the form r ::= [p]v, where r is an r-item involving A.

3. An integer j ϵ {0,...,| σ |} which indicates the number of elements in σ that have been scanned so far and is used as a pointer to the (j+1)-th s-item in σ, denoted by p(j+1). When j = | σ |, s is a final state, since all s-items in p have been scanned.

4. A pointer f, back to the state from which the application of the current s-production p, has been attempted.

5. An r-word w_s, with some marked s-items and r-items.

Let us now discuss about some notations used in the parsing algorithm. Let s = <p,Rs,j,f,w_s> be a state, and x be a terminal s-item appearing in p. For any terminal s-item y appearing in p or on l.h.s. of some production (r-production) in Rs, denote by R_{xy}(p) and R_{xy}(Rs), the sets of terminal r-items involving x and y in p and Rs respectively. NR(s,x) will be used to denote set of non-terminal r-items involving x on r.h.s. of p and on r.h.s. of r-production in Rs. If s ϵ X_i is a state then D(s), which is a set of states, is defined inductively as (i) s ϵ D(s) (ii)if f ϵ X_i then f ϵ D(s).

The parsing procedure is informally discussed in the next paragraph.
Initially the set of states X_0 contains only one state s = <p_0,$\emptyset$,0,0,w> where [p_0] = [Z ::=(S)], Z is a new nonterminal symbol, and S is the starting symbol of the grammar. The algorithm applies either predictor, scanner, or completer on the states in order. These operations are discussed below:
The predictor operation is applied to a nonfinal state s = <p,Rs,j,f,w> when p(j+1) is a nonterminal N. All possible derivations are performed. In other words N is expanded using all the s-productions with N as l.h.s. For each possible derivation step, a new state s^1 = <p^1,Rs^1,j^1,f^1,w> is added to X_i such that p^1 = [N ::= ρ^1,σ^1] ϵ P, Rs^1 contains an r-production of the form r ::= [p^1]v for each r-item r ϵ NR(s,N^{j+1}), j = 0, f^1 is set to s. The state s^1 is added to X_i provided it is not already in X_i.
Scanner operation is applied to an nonfinal state s = <p,Rs,j,f,w> when p(j+1) is a terminal 'a'. For any nonmarked occurence a^k of such a symbol in w, let p[a^k] denote the s-production ϵ P and Rs[a^k] denote the set of r-productions ϵ Rs where a is marked as a^k. The scanner verifies that the elements in $R_{a^k y}$[p[a^k]], where y is

already scanned s-item, and the elements in $R_{a^k y}[Rs[a^k]]$ for all y ϵ l.h.s. of some r-production ϵ Rs, are the nonmarked r-items in w. If this condition is verified the state <p[a^k],Rs[a^k],j+1,f,w$\sim$> is added to the set of states X_{i+1}, where w$\sim$ is obtained from w by marking the s-item a^k and the r-items that were scanned in p[a^k] and Rs[a^k]. If X_{i+1} is empty after completion of scanner operation on x_i, then the input is rejected(erroneous input).

The completer is applied to a final state s = <p,Rs,j,f,w> in X_i. It causes the parser to go back to the state pointed by f, let us say <p^1,Rs^1,j^1,f^1,w^1>, and brings it to X_i as <p^1,Rs^1,j^1+1,f^1,w>.

If a state <p_0,$\emptyset$,1,0,w> is reached in X_i then input sentence is accepted. The procedure is formalised in Figure 3.5.

Initialization: X_0 contains the state s = <[Z ::= (S),∅,0,w,0>,
For i = 0 **to** n **do**
{Process the states of X_i in order, performing one of the following three operations on each state s = <p,Rs,j,f,w_s>
/* predictor phase */
if s is nonfinal and p(j+1) = N ∈ V_N **then**
{Create all the states of the form s^1 = <p^1,Rs^1,0,s,w_s> where p^1 = [N ::= σ,v] and Rs^1 contains an r-production of the form r ::= [p^1]ρ, for each r-item r ∈ NR(s,N^{j+1})
If there is no state of the form <p^1,Rs^1,0,w_s> in D(s) **then** add s^1 to X_i}.
/* scanning phase */
if s is nonfinal and p(j+1) = a ∈ V_T **then**
{ **For** each non-marked occurrence a^k of the symbol a in w_s
If $R_{a^k y}$(p[a^k]) contains all the non-marked r-items in w_s that involve a^k and y for all y = p(h) with 1≤h≤j **and** $R_{a^k y}$(Rs[a^k]) contains all the non-marked r-items in w_s that involve a^k and y for all y = left(τ) with τ ∈ Rs **then**
Add to the set of states X_{i+1} the state <p[a^k],Rs[a^k],j+1,f,w_s,~>}
/* completer phase */
if s is final **then**
go back to the state f = <p^1,Rs^1,j^1,f,$w_s{}^1$> and add to X_i the state <p^1,Rs^1,j^1+1,f^1,$w_s{}^1$>
if X_{i+1} is empty **then** reject
}
If X_n contains a state s = <[Z ::= (S)],∅,1,0,w> such that all the items in w are marked, **then** the input sentence is accepted, otherwise it is rejected.

Figure 3.5: Parsing algorithm based on 1NS-RG.

The parsing algorithm is applied to the r-word w = <{ww^1,ww^2,cw^1,tb^1,tb^2}, {ins(ww^1,tb^1),right(tb^1,ww^2),ins(ww^2,tb^2),sca(tb^2,ww^2),right(tb^2,cw^1),sca(cw^1,tb^2)}>, which corresponds to Figure 3.4.

The steps involved in parsing are as follows:

Initially the set X_0 contains the state 0 = <[Z ::= (S)],∅,0,0,w>. Predictor is applied to this state and the following states are added to X_0.

1 = <[S ::= (tb^1,W^2,S^3){ins(W^2,tb^1),right(tb^1,S^3)}],∅,0,0,w>,

2 = <[S ::= (cw^1,W^2){right(W^2,cw^1),sca(cw^1,W^2)}],∅,0,0,w>,

3 = <[S ::= (tb^1,cw^2,W^3){ins_sca(W^3,tb^1),right(tb^1,cw^2),sca(cw^2,tb^1)}],∅,0,0,w>}.

The scanner is applied to the states 1, 2, and 3; a new state X_1 is created containing the states:

4 = <p_4,∅,1,0,w_4>, with p_4 = [S ::= (tb^1,W^2,S^3){ins(W^2,tb^1),right(tb^1,S^3)}]

w_4 = <{ww^1,ww^2,cw^1,tb^1,tb^2},{ins(ww^1,tb^1),right(tb^1,ww^2),
ins(ww^2,tb^2),sca(tb^2,ww^2),right(tb^2,cw^1),sca(cw^1,tb^2)}>

5 = <p_5,∅,1,0,w_5>, with p_5 = [S ::= (tb^2,W^2,S^3){ins(W^2,tb^2),right(tb^2,S^3)}]

w_5 = <{ww^1,ww^2,cw^1,tb^1,tb^2},{ins(ww^1,tb^1),right(tb^1,ww^2),ins(ww^2,tb^2),
sca(tb^2,ww^2),right(tb^2,cw^1),sca(cw^1,tb^2)}>

$6 = <p_6,\emptyset,1,0,w_6>$, with p_6 = [S ::= ($\mathbf{cw}^1$,W^2){right(W^2,$\mathbf{cw}^1$),sca($\mathbf{cw}^1$,W^2)}]

w_6 = <{ww^1,ww^2,$\mathbf{cw}^1$,$\mathbf{tb}^1$,$\mathbf{tb}^2$},{ins(ww^1,$\mathbf{tb}^1$),right($\mathbf{tb}^1$,ww^2),ins(ww^2,$\mathbf{tb}^2$),
sca($\mathbf{tb}^2$,ww^2),right($\mathbf{tb}^2$,$\mathbf{cw}^1$),sca($\mathbf{cw}^1$,$\mathbf{tb}^2$)}>

$7 = <p_7,\emptyset,1,0,w_7>$, with p_7 = [S ::= (($\mathbf{tb}^1$,$\mathbf{cw}^2$,W^3){ins_sca(W^3,$\mathbf{tb}^1$),right($\mathbf{tb}^1$,$\mathbf{cw}^2$),
sca($\mathbf{cw}^2$,$\mathbf{tb}^1$)}]

w_7 = <{ww^1,ww^2,$\mathbf{cw}^1$,$\mathbf{tb}^1$,$\mathbf{tb}^2$},{ins(ww^1,$\mathbf{tb}^1$),right($\mathbf{tb}^1$,ww^2), ins(ww^2,$\mathbf{tb}^2$),
sca($\mathbf{tb}^2$,ww^2),right($\mathbf{tb}^2$,$\mathbf{cw}^1$),sca($\mathbf{cw}^1$,$\mathbf{tb}^2$)}>

$8 = <p_8,\emptyset,1,0,w_8>$, with p_8 = [S ::= (($\mathbf{tb}^2$,$\mathbf{cw}^2$,W^3){ins_sca(W^3,$\mathbf{tb}^2$),right($\mathbf{tb}^2$,$\mathbf{cw}^2$),
sca($\mathbf{cw}^2$,$\mathbf{tb}^1$)}]

w_8 = <{ww^1,ww^2,$\mathbf{cw}^1$,$\mathbf{tb}^1$,$\mathbf{tb}^2$},{ins(ww^1,$\mathbf{tb}^1$),right($\mathbf{tb}^1$,ww^2),ins(ww^2,$\mathbf{tb}^2$),
sca($\mathbf{tb}^2$,ww^2),right($\mathbf{tb}^2$,$\mathbf{cw}^1$),sca($\mathbf{cw}^1$,$\mathbf{tb}^2$)}>

The predictor applied to states 4, 5, and 6, adds the following states to X_1:

$9 = <p_9,Rs_9,0,4,w_9>$, p_9 = [W ::= (ww^1)]

Rs_9 = {ins(W,$\mathbf{tb}^1$) ::= [4]{ins(ww^1,$\mathbf{tb}^1$)}}

w_9 = <{ww^1,ww^2,$\mathbf{cw}^1$,$\mathbf{tb}^1$,$\mathbf{tb}^2$},{ins(ww^1,$\mathbf{tb}^1$),right($\mathbf{tb}^1$,ww^2),ins(ww^2,$\mathbf{tb}^2$),
sca($\mathbf{tb}^2$,ww^2),right($\mathbf{tb}^2$,$\mathbf{cw}^1$),sca($\mathbf{cw}^1$,$\mathbf{tb}^2$)}>

$10 = <p_{10},Rs_{10},0,5,w_{10}>$, p_{10} = [W ::= (ww^1)]

Rs_{10} = {ins(W,$\mathbf{tb}^2$) ::= [4]{ins(ww^1,$\mathbf{tb}^2$)}}

w_{10} = <{ww^1,ww^2,$\mathbf{cw}^1$,$\mathbf{tb}^1$,$\mathbf{tb}^2$},{ins(ww^1,$\mathbf{tb}^1$),right($\mathbf{tb}^1$,ww^2),ins(ww^2,$\mathbf{tb}^2$),

sca(tb^2,ww^2),right(tb^2,cw^1),sca(cw^1,tb^2)}>

11 = <p_{11},∅,0,6,w_{11}>, p_{11} = [W ::= (ww^1)]

Rs_{11} = {right(W,cw^1) ::= [4]{right(ww^1,cw^1)}, sca(cw^1,W) ::= [4]sca(cw^1,ww^1),}

w_{11} = <{ww^1,ww^2,cw^1,tb^1,tb^2},{ins(ww^1,tb^1),right(tb^1,ww^2),ins(ww^2,tb^2),

sca(tb^2,ww^2),right(tb^2,cw^1),sca(cw^1,tb^2)}>

The scanning phase applied to states 7, 8, 9, 10 and 11, selects state 9 and marks the s-item ww^1, since ins(ww^1,tb^1) is the only r-item involving ww^1 and tb^1 in the input sentence. The resulting state is added to X_2:

12 = <p_{12},Rs_{12},1,4,w_{12}>, p12 = [W ::= (ww^1)]

Rs_{12} = {ins(W,tb^1) ::= [4]{ins(ww^1,tb^1)}}

w_{12} = <{ww^1,ww^2,cw^1,tb^1,tb^2},{ins(ww^1, tb^1),right(tb^1,ww^2),ins(ww^2,tb^2),

sca(tb^2,ww^2),right(tb^2,cw^1),sca(cw^1,tb^2)}>

Since state 12 is a final state completer is applied. It adds to X_2 the state:

13 = <p_{13},∅,2,0,w_{13}>, p13 = [S ::= (tb^1,W^2,S^3){ins(W^2,tb^1),right(tb^1,S^3)}] and

w_{13} = <{ww^1,ww^2,cw^1,tb^1,tb^2},{ins(ww^1, tb^1),right(tb^1,ww^2),ins(ww^2,tb^2),

sca(tb^2,ww^2),right(tb^2,cw^1),sca(cw^1,tb^2)}>

The parsing process continues like this and at the end of sixth iteration the scanner adds to X_5 the state:

24 = <p_{24},Rs_{24},1,22,w_{24}>, p_{24} = [W ::= (ww^2}]

Rs_{24} = {ins_sca(W,tb^2) ::= [4]{ins(ww^2,tb^2),sca(tb^2,ww^2)},

right(tb^1,W) ::= [4]{right(tb^1,ww^2)}

w_{24} = <{ww^1,ww^2,cw^1,tb^1,tb^2},{ins(ww^1, tb^1),right(tb^1,ww^2),ins(ww^2,tb^2),

sca(tb^2,ww^2),right(tb^2,cw^1),sca(cw^1,tb^2)}>

Since this is a final state, the completer adds to X_5:

25 = <p_{25},Rs_{25},3,13,w_{25}>, p_{25} = [S ::= ((tb^2,cw^1,W^3){ins_sca(W,tb^2),right(tb^2,cw^1),

sca(cw^1,tb^2)}]

Rs_{25} = {right(tb^1,S) ::= [3]{right(tb^1,W)}}

w_{25} = w_{24}

26 = <p_{26},∅,3,0,w_{24}>, p_{26} = [S ::= (tb^1,W,S),{ins(W,tb^1),right(tb^1,S)}]

27 = <p_0,∅,1,0,w_{24}>, p_0 = [Z ::= (S)]

Since X_5 contains the state 27 = <p_0,∅,1,0,w> such that all the items in w are marked, the input sentence is accepted.

This algorithm, in general, has exponential time complexity. But when applied to languages having additional properties of connection and degree-boundedness, such as diagrammatic languages, it has polynomial time complexity. As it is evident, the drawback of this algorithm is its lack of efficiency. It is more suitable for diagrammatic languages but not iconic languages. Further it is only a recognition algorithm not a complete parsing algorithm. It does not produce the parse tree.

Chapter 4

Modified Relation Grammar/1 (MRG/1)

4.1 Modified Relation Grammar/1 (MRG/1)

Modified Relation Grammar/1 (MRG/1 for short) is another restricted form of Relation Grammar. It is obtained by restricting the form of productions, and distributing the evaluation rules among various productions of the production set. A sentence in MRG/1 is represented as a pair (T,RT), where T is a multiset of terminal symbols and RT is a set of relation instances over those symbols. The symbols in T and consequently the constraints in RT are totally ordered. They are ordered according to the ordering of spatial relations in the set V_R. As in case of RG/1, here also we assume that the set of relations in V_R are totally ordered. Let us look at the ordering of symbols of input sentence in some detail.

Let $r_1, r_2, \ldots, r_n$ be the spatial relations belonging to the set V_R, with ordering $r_1 < r_2 < \ldots < r_n$. The symbols of T are arranged as follows:

Initially T is empty. First we start at an initial position, which is the minimum position of all relations $r_1, r_2, \ldots, r_n$. We add the symbol at this position to T. Next we add all the symbols among the immediate neighbours of this symbol, linked to it via the r_1 relation, to the set T. Then we add all the symbols among its immediate neighbours, linked to it via r_2 relation, to the set T; and so on. Once all the symbols linked to the first symbol in T via the r_n relation are added, we go to the second symbol in T. For the second symbol in T we add all the symbols among its immediate neighbours, which are linked to it via the r_1 relation, to the set T. These symbols will be added to the set T provided they are not already included in the set T. Next we find all the symbols among immediate neighbours of second symbol in T, linked to it via r_2 relation. We add these symbols to the set T, if they are already not in T. We repeat this procedure for the third, fourth, fifth and so on, symbols in T; until all the symbols of the input sentence are added to the set T.
The constraints part of the input sentence i.e. the set RT is ordered as follows:
First we add all the constraints involving the first symbol in T and its immediate neighbours, to the set RT. Then we add all the constraints involving the second symbol in T and its immediate neighbours, to the set RT. Only those constraints involving the second symbol in T which are already not added to RT, will be added now. Then we add all the constraints involving the third symbol in T and its immediate neighbours; and so on. This procedure is repeated until all the constraints of the input sentence are added to the set RT. In this ordering we implicitly assumed that a constraint is specified only over two immediately adjacent icons.
To clarify the ordering of input sentences, let us consider a couple of sample iconic sentences.

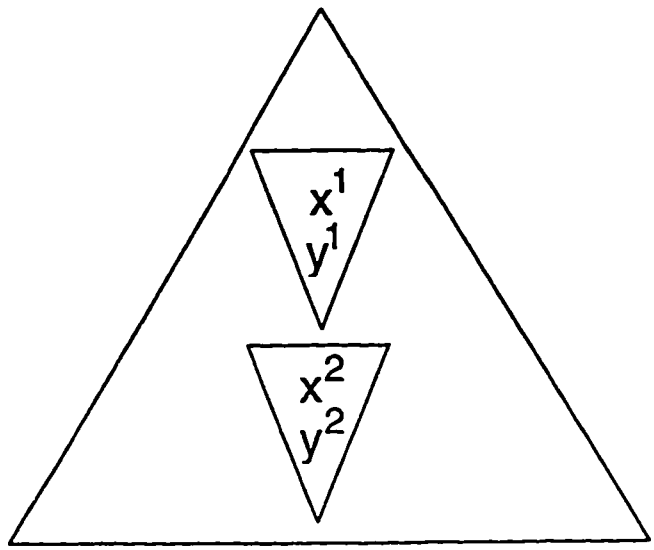

Figure 4.1: A sample iconic sentence.

Example 4.1: Assuming the set of relations V_R to be {enclose, above}, and ordering of relations to be enclose<above. The iconic sentence[1] in Figure 4.1 can be ordered as w = (T,RT) = ({Δ^1,∇^1,∇^2,x^1,y^1,x^2,y^2},{enclose(Δ^1,∇^1), enclose(Δ^1,∇^2),enclose(∇^1,x^1),enclose(∇^1,y^1),above(∇^1,∇^2),enclose(∇^2,x^2), enclose(∇^2,y^2),above(x^1,y^1),above(x^2,y^2)}.

Example 4.2: Assuming set of relations to be V_R = {x,y,z}, and x<y<z to be their ordering. The iconic sentence

v = $\clubsuit^1$ $\clubsuit^2$ $\spadesuit^2$

$\spadesuit^1$ $\spadesuit^3$ $\spadesuit^4$

[1]This sentence is similar to a Vennlisp [22] sentence, except for the shape of enclosing polygons

can be ordered as v = (T,RT) = ({♣^{1},♣^{2},♠^{1},♠^{2},♠^{3},♠^{4}}{x(♣^{1},♣^{2}),y(♣^{1},♠^{1}), x(♣^{2},♠^{2}),y(♣^{2},♠^{3}),z(♣^{2},♠^{1}),x(♠^{1},♠^{3}),y(♠^{2},♠^{4}),z(♠^{2},♠^{3}),x(♠^{3},♠^{4})}).

In the parsing algorithm to be presented in chapter 5, we assume that the input sentences are totally ordered as described above. We assume that this ordering is performed by the lexical analyzer. Now let us look at the MRG/1 formalism in detail with examples. First we begin with the formal definition of MRG/1.

Definition 2 *A Modified Relation Grammar/1 (MRG/1 for short) is defined as a 5-tuple* $G = (V_N, V_T, V_R, S, P)$ *where*

- V_N *is a finite set of nonterminal symbols*
- V_T *is a finite set of terminal symbols*
- V_R *is a finite set of spatial relation symbols,* $V_R = \{r_i | 1 \leq i \leq k\}$. *For each relation symbol* r_i *the function* $a: V_R \rightarrow Z$, *where Z is the set of postive integers, is defined.*
- *S is the starting symbol*
- *P is a finite set of productions, where each production is either of the following two types:*

  1. $A := \{y, Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m}\}\{r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), \ldots, r_m(y, Y_m^{l_m}), R_1, R_2, \ldots, R_k\}$ $\{E_1, E_2, \ldots, E_n\}$ *where*

$K \geq 0$; $n \geq 0$; $y \in V_T$; $A \in V_N$; $Y_i^{l_i}$, $i=1,...,m$, $\in V_N$. The superscripts $l_1, l_2, ..., l_m$ on Y_is denotes the occurence of the symbol on r.h.s. of production. If a nonterminal symbol $Y_i^{l_i}$ has only one occurence on r.h.s., then it is superscripted as one. If there are two or more occurences of a symbol Y_i, then different occurences are superscripted as 1,2,3,.... and so on. $r_h \in V_R$, $h=1,...,m-1$; for any $i,j \in \{1,...,m\}$ r_i, r_j need not be distinct. $R_1, R_2, ..., R_k$ denotes possible additional constraints over the symbols $Y_i^{l_i}$, $i=1,...,m$. $E_1, E_2, ..., E_n$ denotes the external constraints of the production. These are nothing but evaluation rules associated with the production. They specify how the symbols $\{y, Y_1^{l_1}, Y_2^{l_2}, ..., Y_m^{l_m}\}$ should be related to neighbours of A, when A is replaced by $\{y, Y_1^{l_1}, Y_2^{l_2}, ..., Y_m^{l_m}\}$ during a derivation step.

2. *$A := \{y\}\{\}\{E_1, E_2, ..., E_n\}$ where*

   $A \in V_N$; $y \in V_T$; $E_1, E_2, ..., E_n$ denotes the external constraints of the production.

Intuitively type 1 production means the following:
Nonterminal symbol A, on input y, is to be replaced by the set $\{y, Y_1^{l_1}, Y_2^{l_2}, ..., Y_m^{l_m}\}$, provided the constraints $\{r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), ..., r_m(y, Y_m^{l_m}), R_1, R_2, ..., R_k\}$ are satisfied. In a derivation, a constraint $r(u_1, ..., u_h)$ is satisfied if all $u_i (1 \leq i \leq j) \in V_T$. If some $u_i (1 \leq i \leq j) \in V_N$ then external constraints are used to reduce the constraint $r(u_1, ..., u_h)$, at later stages of derivation. This process becomes clear when we discuss the parsing algorithm. The set of constraints $\{r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), ..., r_m(y, Y_m^{l_m}), R_1, R_2, ..., R_k\}$ are called internal constraints. In internal constraints, the constraints $r_1(y, Y_1^{l_1}), r_2(y, Y_2^{l_2}), ..., r_m(y, Y_m^{l_m})$ are compulsory. The constraints $R_1, R_2, ..., R_k$ are optional, they denote

possible additional constraints among the symbols $Y_1^{l_1}, Y_2^{l_2}, \ldots, Y_m^{l_m}$. If a production of type 1 is applied, then an occurence of symbol on l.h.s. of production will be replaced by the set of symbols on r.h.s. of production, i.e., A will be replaced by the set $(y, Y_1^{l_1}, Y_2^{l_2}, Y_m^{l_m})$. These symbols will be arranged according to the internal constraints. The set $\{E_1, E_2, \ldots, E_n\}$ denotes another set of constraints called the external constraints. These are nothing but the evaluation rules. They specify how the symbols of r.h.s. of production are to be connected to the neighbours of the symbol on l.h.s. of production, in a derivation tree, when the production is applied. An external constraint has the form

$rel(x_1, \ldots, x_k)$:- [constr]

where [constr] $= \{p_1, \ldots, p_h\}$ is the set of constraints. The relation instance $rel(x_1, \ldots, x_k)$ on the l.h.s. of an external constraint has to be internal constraint of some MRG/1 production. The constraints [constr] specifies how the constraint $rel(x_1, \ldots, x_k)$ will be reduced when the current production is applied. By reduction of constraint we mean how the constraint $rel(x_1, \ldots, x_k)$ is going to be changed when the current production is applied. The first symbol on r.h.s. of a production (i.e., the symbol denoted by y) is called the primary symbol.

Type 2 production states that the nonterminal A, on input y, is to be replaced by y. There are no internal constraints. The external constraints $E_1, E_2, \ldots, E_n$ has the same meaning as in type 1 production. y in this case also, is the primary symbol.

Further the following condition has to be satisfied by any MRG/1 production set.

- No two productions having the same l.h.s. should have the same primary

symbol

Let us now consider some examples to further clarify the definition.

Example 4.3: Let us give MRG/1 description of a language whose sentences consists of 2D grids of two kinds of trees. Trees of type ♠ are placed on the South-East frontier, remaining grid positions are filled by ♣ type of trees. A sample sentence of this language was shown in example 4.2. It is reproduced below for sake of convenience.

v = $\clubsuit^1$ $\clubsuit^2$ $\spadesuit^2$
$\spadesuit^1$ $\spadesuit^3$ $\spadesuit^4$

The MRG/1 for this language is written as G := (V_N,V_T,V_R,S,P) := ({S,B,A},{♣,♠},{x,y,z},S,P) where P consists of

1. S := {♣,B^1,A^1} {x(♣,B^1),y(♣,A^1),z(B^1,A^1)}
2. B := {♣,B^1,A^1} {x(♣,B^1),y(♣,A^1),z(B^1,A^1)} {x($\clubsuit^m$,B) :- x($\clubsuit^m$,♣); z(B,A^m) :- z(♣,A^m),x(A^m,A^1)}
3. B := {♠,A^1} {y(♠,A^1)} {x($\clubsuit^m$,B) :- x($\clubsuit^m$,♠); z(B,A^m) :- z(♠,A^m),x(A^m,A^1)}
4. A := {♣,A^1} {y(♣,A^1)} {y($\clubsuit^m$,A) :- y($\clubsuit^m$,♣); z(A,A^m) :- z(♣,A^m),x(A^m,A^1); z($\spadesuit^m$,A) :- z($\spadesuit^m$,♣); z($\clubsuit^m$,A) :- z($\clubsuit^m$,♣); x(A,A^m) :- x(♣,A^m),z(A^m,A^1); x($\clubsuit^m$,A) :- x($\clubsuit^m$,♣)}
5. A := {♠} {} {y($\spadesuit^m$,A) :- y($\spadesuit^m$,♠); y($\clubsuit^m$,A) :- y($\clubsuit^m$,♠); z($\spadesuit^m$,A) :- z($\spadesuit^m$,♠);

$z(\clubsuit^m,A)$:- $z(\clubsuit^m,\spadesuit)$; $z(A,A^m)$:- $z(\spadesuit,A^m)$; $x(A,A^m)$:- $x(\spadesuit,A^m)$; $x(\spadesuit^m,A)$:- $x(\spadesuit^m,\spadesuit)$; $x(A^m,A)$:- $x(A^m,\spadesuit)$}

Example 4.4: Let us give MRG/1 description of the iconic sentence shown in Figure 4.1. MRG/1 description for that sentence[2] w, can be written as $G := (V_N,V_T,V_R,S,P) := (\{S,A,B\},\{\triangle,\triangledown,x,y\},\{enclose,above\},S,P)$ where P consists of

1. $S := \{\triangle,A^1,A^2\}$ {enclose($\triangle$,A^1),enclose($\triangle$,A^2),above(A^1,A^2)}

2. $A := \{\triangledown,B^1,B^2\}$ {enclose($\triangledown$,B^1),enclose($\triangledown$,B^2),above(B^1,B^2)} {enclose($\triangle^m$,A) :- enclose($\triangle^m$,$\triangledown$); above(A^m,A) :- above(A^m,$\triangledown$); above(A,A^m) :- above($\triangledown$,A^m); above($\triangledown^m$,A) :- above($\triangledown^m$,$\triangledown$); above(A,$\triangledown^m$) :- above($\triangledown$,$\triangledown^m$);}

3. B := {x} {} {enclose($\triangledown^m$,B) :- enclose($\triangledown^m$,x); above(B^m,B) :- above(B^m,x); above(B,B^m) :- above(x,B^m)}

4. B := {y} {} {enclose($\triangledown^m$,B) :- enclose($\triangledown^m$,y); above(B^m,B) :- above(B^m,y); above(B,B^m) :- above(y,B^m); above(x^m,B) :- above(x^m,y)}

Superscript m on the symbols in external constraints of the preceeding examples denote an arbitrary occurence of the symbol. From the definition and examples it is clear that MRG/1 is very much similar to the RG/1 grammar. The comparision of these two formalisms is given in section 4.2. Before going to that section let us discuss about the language defined by the MRG/1 formalism.

[2]This description can be extended to describe other similar sentences

4.1.1 Language Defined by MRG/1

We will now formally define the language generated by MRG/1 G = (V_N,V_T,V_R,S,P). To do so first we will define what it means by a sentential form.

Definition 3 *A pair (T^1,RT^1), where T^1 is a multiset of symbols from $V_N \cup V_T$, and RT^1 is a set of spatial relations in V_R over symbols of T^1, is said to be a sentential form of MRG/1, if and only if there exists a derivation sequence*
$(\{S\},\emptyset) \Rightarrow (T_1^1,RT_1^1) \Rightarrow (T_2^1,RT_2^1) \Rightarrow \ldots \Rightarrow (T_n^1,RT_n^1) \Rightarrow (T^1,RT^1)$.

In the above derivation sequence all the intermediate sentential forms are totally ordered, and the leftmost nonterminal is expanded at each step. In other words, the sentential form (T^1,RT^1) is obtained from ({S},$\emptyset$) by constructing the derivation tree in a top down breadth first manner. Only these type of derivations will be permitted in an MRG/1 grammar. This is because, if a production is applied all its constraints have to be satisfied. All the constraints of a production can be satisfied only when all its children are expanded in a breadth first manner. Now we will define language generated by the MRG/1.

Definition 4 *Given a MRG/1 Grammar $G = (V_N, V_T, V_R, S, P)$, the language defined by G, denoted by L(G), is the set of all sentential forms (T,RT), where T consists of terminals only. In other words,*
$L(G) = \{(T,RT) | (\{S\},\emptyset) \stackrel{*}{\Rightarrow} (T,RT)$ *and T consists of terminal symbols only*$\}$.

4.2 Comparision of MRG/1 and RG/1 Formalisms

Although MRG/1 and RG/1 are similar in many aspects, they differ in the following ways.

1. In MRG/1, the evaluation rules are associated with the productions as external constraints. In RG/1 grammar, the evaluation rules are stored in a separate set R. The association of evaluation rules with the productions, in MRG/1, helps in improving the clarity of productions. It is clear how the symbols on r.h.s. of a production should be linked to the neighbours of the symbol on the l.h.s. of a production, in a derivation tree, when the production is applied. In RG/1 formalism the evaluation rules are maintained in a separate set R and hence it is not clear how the symbols on r.h.s. of a production should be linked to the neighbours of symbol on l.h.s. of production, in a derivation tree, when the production is applied. In MRG/1 the evaluation rules are distributed among all the productions except the first production (production having starting symbol S as l.h.s.). Due to the distribution of evaluation rules in MRG/1 the time taken to apply a production during parsing is reduced. This is explained in chapter 5.

2. In MRG/1 we have two types of productions, whereas in RG/1 we have only one type of production.

3. In type 1 production of MRG/1, the symbols on r.h.s. other than primary symbol should be nonterminals. In RG/1 productions these symbols can be terminals or nonterminals. This factor however does not reduce the expressiveness of MRG/1 because, for each terminal on r.h.s. of an RG/1 production,

other than primary symbol, we can create a nonterminal and use it in place of the terminal, in r.h.s. of corresponding MRG/1 production. Of course we have to add a type 2 production having the newly created nonterminal as l.h.s. and the terminal symbol as r.h.s. Example 4.5 will clarify this procedure.

Example 4.5: If we have an RG/1 production

$A := \{a,Y_1,Y_2,b\}\ \{x(a,Y_1),y(a,Y_2),z(a,b)\}$

where $A, Y_1, Y_2 \in V_N$ and $a,b \in V_T$.

The corresponding MRG/1 production set can be written as

$A := \{a,Y_1^1,Y_2^1,Y_3^1\}\ \{x(a,Y_1^1),y(a,Y_2^1),z(a,Y_3^1)\}\ \{\}$

$Y_3 := \{b\}\ \{\}\ \{z(a,Y_3)$:- $z(a,b)\}$

where Y_3 is the newly created nonterminal.

It is clear from example 4.5 that the above type of substitution increases the sizes of production and nonterminal sets. Since the sizes of nonterminal and production sets increase just by a constant amount, the asymptotic time complexity of the MRG/1 based parsing algorithm (to be discussed in next chapter) will not be affected (increased). A procedure for converting a generic RG/1 grammar to MRG/1 grammar is given in lemma 4.1.

4. In type 1 production of MRG/1 the primary symbol can be related to all other symbols on r.h.s. by arbitrary relation symbols. These relation symbols need not be distinct. In RG/1 grammar the primary symbol is to be related to all the other symbols of r.h.s. via distinct relation symbols. This restriction reduces the clarity of RG/1 grammars. Sentences such as the one shown in

Figure 4.1 cannot easily be expressed using RG/1 grammar.

So in a nutshell we can say that MRG/1 grammars are similar to, and more clear than, RG/1 grammars. Parsing algorithm based on MRG/1 is described in the next chapter. Now we will compare the expressiveness of MRG/1 and RG/1 formalisms.

4.2.1 Expressiveness of MRG/1 and RG/1 Formalisms

We will show that MRG/1 formalism is equivalent to RG/1 formalism in terms of expressiveness. To prove this, first we will give a procedure which will convert any generic RG/1 formalism to MRG/1 formalism. This establishes that MRG/1 is atleast as expressive as RG/1 formalism. Then we will prove that the language accepted by an RG/1 formalism is also accepted by the corresponding MRG/1 formalism (MRG/1 formalism obtained by converting the RG/1 formalism).

Lemma 4.1: MRG/1 grammar is atleast as expressive as RG/1 grammar.

Proof: To prove this we will give a procedure called 'Convert' which will convert a given RG/1 grammar to MRG/1 grammar. The input to the procedure is an RG/1 grammar G = (V_N,V_T,V_R,S,P,R) and output is an MRG/1 grammar G^1 = ($V_N{}^1$,$V_T{}^1$,$V_R{}^1$,S^1,P^1). The procedure is given below:

Procedure Convert(G,G^1)

1. $V_T{}^1 := V_T$.
2. $V_R{}^1 := V_R$.
3. $S^1 := S$.
4. For each RG/1 production pϵP of the form

$A := \{y,Y_1,Y_2,...,Y_m\}\{r_1(y,Y_1),r_2(y,Y_2),...,r_m(y,Y_m),R_1,R_2,...,R_k\}$

We construct corresponding MRG/1 production/productions as follows:

- If all Y_is i=1,...,m are non terminals then construct an MRG/1 production as

  $A := \{y,Y_1^{l_1},Y_2^{l_2},...,Y_m^{l_m}\}\{r_1(y,Y_1^{l_1}),r_2(y,Y_2^{l_2}),...,r_m(y,Y_m^{l_m}),R_1,R_2,...,R_k\}$

  $\{E_1,E_2,...,E_n\}$

  where $l_1,l_2,...,l_m$ denote different occurences of symbols $Y_1,Y_2,...,Y_m$ on r.h.s. of production. $E_1,E_2,...,E_n$ are the external constraints of the MRG/1 production. Add this production to the set P^1. Add A, $Y_1,Y_2,...,Y_m$ to the set $V_N{}^1$. The external constraints $E_1,E_2,...,E_n$ are nothing but evaluation rules of the corresponding RG/1 grammar, of the form

  $t_1(X_1,A)$:- $(A \Rightarrow \{y,Y_1,Y_2,...,Y_m\}),q_1(X_1,Z_1)$

  .

  .

  .

  $t_n(X_n,A)$:- $(A \Rightarrow \{y,Y_1,Y_2,...,Y_m\}),q_n(X_n,Z_n)$

  where $Z_j \in y \cup \{Y_1,Y_2,...,Y_m\}$, $q_j \in V_R$, for j=1,...,n.

  These evaluation rules are rewritten as

  $t_1(X_1,A)$:- $q_1(X_1,Z_1)$

  .

  .

  .

  $t_n(X_n,A)$:- $q_n(X_n,Z_n)$

  and associated with the MRG/1 production as external constraints.

- If some Y_i, let us say Y_k, $k \in \{1,...,m\}$, is a terminal then we create a nonter-

minal W_k and an MRG/1 production

$W_k := \{Y_k\}\{\}\{E_1,E_2,...,E_n\}$ where $E_1,E_2,...,E_n$ are external constraints. These constraints are of the form

$t_1(X_1,W_k)$:- $t_1(X_1,Y_k)$

.

.

.

$t_n(X_1,W_k)$:- $t_n(X_1,Y_k)$

where $t_1(X_1,Y_k),...,t_n(X_1,Y_k)$ are constraints involving Y_k on the r.h.s. of RG/1 production. Add W_k to the set $V_N{}^1$, and the production $W_k := \{Y_k\}\{\}\{E_1,E_2,...,E_n\}$ to the set P^1. We create another MRG/1 production (which actually corresponds to the given RG/1 production $p \epsilon P$) of the form

$A := \{y,Y_1^{l_1},Y_2^{l_2},...,W_k^{l_k},...,Y_m^{l_m}\}$

$\{r_1(y,Y_1^{l_1}),r_2(y,Y_2^{l_2}),...,r_k(y,W_k^{l_k}),...,r_m(y,Y_m^{l_m}),R_1,R_2,...,R_k\}$

$\{E_1,E_2,...,E_n\}$

where $l_1,l_2,...,l_m$ denotes different occurences of symbols $Y_1,Y_2,...,Y_m$ on r.h.s. of production. $E_1,E_2,...,E_n$ are the external constraints of the MRG/1 production. Add this production to the set P^1. Add A, $Y_1,Y_2,...,Y_m$ (if they are already not added) to the set $V_N{}^1$. The external constraints in this case are nothing but evaluation rules of the corresponding RG/1 grammar which are rewritten as discussed earlier.

Lemma 4.2: If L = L(G) for some RG/1 grammar G, then L = $L(G^1)$ for the corresponding MRG/1 G^1.

Proof: We will prove this lemma in two parts. First we will prove that all $w \epsilon L(G)$

also belongs to $L(G^1)$. Then we will prove that if a sentence $w \notin L(G)$, then $w \notin L(G^1)$.

Part 1: L(G) for a RG/1 grammar is defined as

$L(G) = \{(T,RT) | (S,\emptyset) \stackrel{*}{\Rightarrow} (T,RT)$ and T consists of terminal symbols only$\}$.

In the derivation of any sentence in G, at any derivation step, only the leftmost nonterminal in the sentential form will be expanded, i.e., the sentence (T,RT) will be obtained by a sequence of derivations,

$(\{S\},\emptyset) \Rightarrow (T_1,RT_1) \Rightarrow (T_2,RT_2) \Rightarrow \ldots \Rightarrow (T_n,RT_n) \Rightarrow (T,RT)$

where at each step leftmost nonterminal in $T_i (1 \leq i \leq n)$ will be expanded. In other words the sentence (T,RT) will be derived from S by developing (expanding) the derivation tree in top down breadth first order.

In the MRG/1 obtained by procedure 'Convert' of lemma 4.1, $S^1 = S$. Further all the leftmost nonterminals expanded at various steps in RG/1, i.e., all the leftmost non terminals in $T_i (1 \leq i \leq n)$, also belongs to V_N^1. Also that there exists productions for these nonterminals in G^1 which has the same r.h.s. as G productions. Apart from this, the derivation in MRG/1 also proceeds in top down breath first manner. From all these facts it is clear that every sentence generated by RG/1 grammar G can also be generated by MRG/1 grammar G^1.

Part 2: Now let us prove that if a sentence $w = (T,RT) \notin L(G)$ then it does not belong to $L(G^1)$.

A sentence $w = (T,RT) \notin L(G)$ if there does not exist a leftmost derivation for it. Such a case is possible when a leftmost nonterminal in a particular sentential

form cannot be expanded. Since $V_N \subseteq V_N^1$, this nonterminal belongs to V_N^1 also. Because the nature of derivation in MRG/1 is same as that of RG/1 (derivation in both RG/1 and MRG/1 proceeds in top down breadth first manner), and $S^1 = S$, such a nonterminal appears in a MRG/1 sentential form also; and it cannot be expanded. So MRG/1 derivation for w is not possible. In other words $w \notin L(G^1)$.

Chapter 5

Parsing Algorithm

5.1 Overview

In this chapter we will describe a table based form of predictive parser for Iconic VPLs based on MRG/1 formalism. The organization of lexical and syntax analysis phases is shown in Figure 5.1. The lexical analyzer takes the iconic pattern as input and converts it into an iconic sentence represented as (T,RT), where T is a multiset of icons (or symbols) and RT is a set of spatial relations among these icons. The set of icons in T and set of constraints in RT (spatial relations) are totally ordered. The ordering of these sets has been discussed in the previous chapter. We assume that the lexical analyzer returns the sentence (T,RT) in totally ordered form. The parser takes this representation of the iconic sentence and produces a parse tree for it, if it is grammatically correct; else prints error message and stops. The parse tree is produced only in a figurative sense, actually the sequence of applied productions is printed. By sequence of applied productions we mean the (row number ,column number) pairs of applied productions.

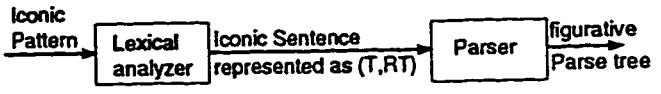

Figure 5.1: Organization of lexical and syntax analysis phases.

In this section, an overview of the parsing method is presented; detailed parsing algorithm is given in the next section.

5.1.1 Parsing Method

The parsing algorithm takes as input an iconic sentence represented as a pair (T,RT), where T is a multiset of icons and RT is a set of relations among these icons. We assume that both the symbols and constraints, of the iconic sentence, are totally ordered. The algorithm uses two data structures R and TQ. List R is used to hold internal constraints of productions applied during parsing, which are not yet resolved. TQ contains all the constraints involving all the symbols considered so far, at a particular step during parsing. Parse table is assumed to be organized as a 2D array M, indexed by the nonterminals (of the grammar) along the rows, and the terminals (of the grammar) along the columns. A production of the form

A := $\{y, Y_1^{l_1}, Y_2^{l_2}, ..., Y_m^{l_m}\}$ {internal constraints} {external constraints}

will be stored in row indexed by A and column indexed by y. Blank entries in the parse table denote errors.

The parsing algorithm works top down and explores the nodes of the parse tree in a breadth first manner. All the nodes at the i^{th} level of the parse tree will be expanded before any node at the $i+1^{th}$ level. At a particular level the nodes will be considered from left to right. At a particular step during parsing, the algorithm tries to find a

production M(A,a), where A is the nonterminal in parse tree to be expanded and 'a' is the current input symbol. If no production is found it reports error and stops. If a production is found, the parser adds the internal constraints of the production to R. The parser then removes all the constraints involving current input symbol 'a' from RT and adds them to TQ. Since the constraints involving 'a' appears towards the begining of RT (because RT is totally ordered), this step can be carried out easily. Then the parser tries to resolve constraints in R involving the current occurence of A. If a constraint is fully resolved (i.e., if it contains only terminal symbols), then it is removed from both R and TQ. The input symbol 'a' is removed from T and the parsing process repeats with the next input symbol and the next parse tree node in the breadth first order.

In brief, the parsing algorithm produces the top down parse of the input sentence as it traverses the parse tree in a breadth first manner. At a particular step during parsing it applies a production and adds internal constraints of the production to R. These constraints will be resolved and satisfied at later steps of parsing. The parsing method is shown in the form of algorithm in Figure 5.2.

Algorithm: The parsing method.

Input: Totally ordered input iconic sentence (T,RT).

Output: Topdown parse of it, if it is correct.

```
Begin

A := S.

a := Head(T).

While(T not empty) do

Begin

  Apply production M(A,a), i.e., print the pair (A,a) if
  M(A,a) not blank; else print error message and stop.

  Add internal constraints of M(A,a) to R.

  Move all constraints involving 'a' from RT to TQ.

  Resolve constraints in R involving current occurence of A.
  Remove fully resolved constraints from R and TQ.

  Dequeue(T).

  A := Next node symbol of parse tree in breadth first order.

  a := Head(T).

end.

Print 'Successful Parsing ' and stop.

end.
```

Figure 5.2: Overview of the Parsing Method.

5.2 MRG/1 Based Predictive Parser

In this section we will see the details of the MRG/1 based parsing algorithm. Schematic form of the parser is shown in Figure 5.3. The iconic sentence to

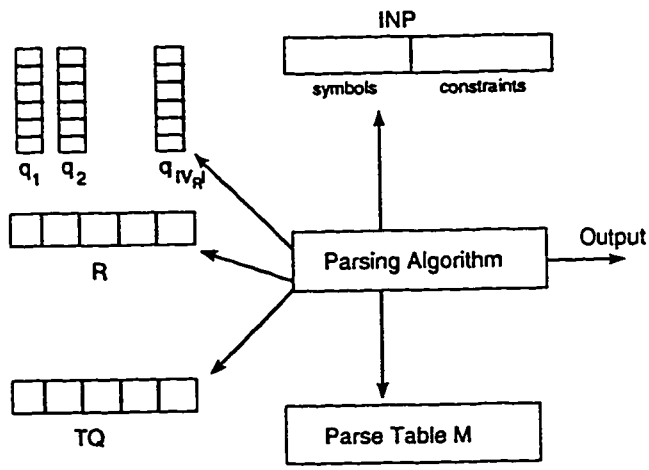

Figure 5.3: MRG/1 based Predictive Parser.

be parsed is stored in an input buffer called INP. It has got two parts, symbols and constraints. The symbols part stores the symbols of input sentence, i.e., the T part; and the constraints part stores the set of constraints of input sentence, i.e., the RT part. The parser uses the following data structures.

1. It uses $|V_R|$ temporary queues, where V_R is the set of spatial relations, and $|V_R|$ denotes its size, i.e., number of relations in the set V_R. It creates these queues at the begining of parsing process. These queues are used to hold the intermediate nodes of the parse tree.

2. It uses a data structure R (R can be organized as an array). This data structure is used to hold internal constraints of applied productions which are not yet resolved.

3. It uses another data structure TQ. This data structure contains all the constraints, involving all the symbols considered so far, at a particular step during parsing.

4. A parse table M, which is organized as a 2D array, indexed by the nonterminals (of the given MRG/1) along the rows and terminals (of the given MRG/1) along the columns. An entry of the parse table could be a blank, or it may store a production of the grammar. Blank entries denote errors. A production of the form

   A := $\{y,Y_1^{l_1},Y_2^{l_1},...,Y_m^{l_m}\}$ {internal constraints} {external constraints}

   will be stored in row indexed by A and column indexed by y, in the parse table.

The parser goes through several configurations during its course of execution. The configuration of the parser consists of contents of queues (q_1,q_2,...,$q_{|V_R|}$), TQ, R, and input buffer INP; at a particular step during parsing. The notion of configuration becomes clear when we apply our parsing algorithm to the examples. The details of the parsing algorithm are described in the next subsection.

5.2.1 Details of the Parsing Algorithm

The parsing algorithm takes as input an iconic sentence represented as a pair (T,RT) and produces a parse tree if the sentence is grammatically correct, else prints error

message and stops. The input iconic sentence is totally ordered and it is stored in input buffer INP. T is stored in symbols part and RT is stored in constraints part, of the buffer. Before getting into the details of the algorithm let us look at some of the notations used in the algorithm.

Operation 'Head' on a queue returns the first element of the queue. 'Enqueue(q,S)' adds S to the end of queue q. 'Dequeue' on a queue removes the first element from the queue. Assignment and equality operators are represented by := and ==, respectively. INP.symbols is used to access the symbols part, and INP.constraints is used to access the constraints part, of the INP buffer. We assume that both these parts are organized as queues.

The algorithm is given in Figures 5.4, 5.5, and 5.6. For the sake of clarity it is reproduced in the form of flowchart in Figures 5.7, 5.8, and 5.9.

```
Begin /* Begining of algorithm */

  Create |V_R| queues numbered 1,2,...,|V_R|;

  Enqueue(q_1,S^0);

  k:=0; i:=1; p:=1; q_no:=1; TQ:={}; R:={};

  While (INP.symbols not empty) do

  Begin

    If (queue numbered q_no not empty) then

      A:=Head(q_no);

      If (superscript on A less than or equal to p) then

        Dequeue(q_no);
        a:=Head(INP.symbols);

        If ( M(A,a) not blank) then

          Apply production M(A,a); print the pair (A,a); Add internal
          constraints to R. Prior to insertion in R, nonterminal symbols
          in the internal constraints are attached superscript i and subscript k+l,
          where l is the occurence of nonterminal on r.h.s. of
          production; primary symbol is attached superscript of a, subscript is not
          attached to the primary symbol;

          Remove constraints involving current input symbol 'a' from INP.constraints
          and add them to TQ;

          Resolve constraints in R where A appears according to the external constraints
          of M(A,a). If constraints involving nonterminals on r.h.s. of M(A,a) gets added
          as a result, then subscript the nonterminals of constraints by k+l and superscript
          by i, before adding to R. If a constraint of the form r(x_1,x_2,...,x_n) where
          r ∈ V_R and x_1,x_2,...,x_n ∈ V_T, appears in R,then find for its occurence in TQ.
          If found then remove it from both R and TQ; else print error message and stop;
```

Figure 5.4: Predictive parsing algorithm for Iconic VPLs based on MRG/1.

```
      Insert symbols of r.h.s. of applied production into queues 1,2,...,|V_R|. The
      symbols are inserted as follows: If X is a symbol on r.h.s. of applied production
      and r(a,X) is the constraint involving primary symbol 'a' and X. Insert
      X^i_{k+l} in the queue numbered 'pos' where 'pos' is the position or r in V_R.
      l is the occurence of symbol on r.h.s. of production;

      If (p!=i) then

        Dequeue(INP.symbols);
        k:=k+m;

      else

        i:=i+1;
        Dequeue(INP.symbols);
        k:=k+m;

      endif;
    else /* else of 'If (M(A,a) not blank)' */

      Print error message and stop;

    endif;

  else /* else of 'If (subscript on A <= p)' */

    q_no:=q_no+1;

    If (q_no <= |V_R|) then

      k:=k+m;

    else

      p:=p+1; i:=i+1; q_no:=q_no+1; k:=k+m;

    endif;

  endif; /* endif of If (subscript on A <= p) */
```

Figure 5.5: Parsing algorithm for MRG/1 contd...

```
    else /* else of 'If (queue numbered q_no not empty' */
      If (All queues empty) then
        Print error message and stop;
      else
        q_no:=q_no+1;
        If (q_no <= |V_R|) then
          k:=k+m;
        else
          p:=p+1; i:=i+1; q_no:=1; k:=k+m;
        endif;
      endif;
    endif; /* endif of 'If (queue numbered q_no not empty)' */
  end; /* end of main while loop */
  If (All queues empty .AND. R is empty .AND. TQ is empty
  .AND. INP.constraints is empty) then
    Print succesful parsing and stop;
  else
    Print error message and stop;
  endif;
end. /* end of algorithm */
```

Figure 5.6: Parsing algorithm for MRG/1 contd...

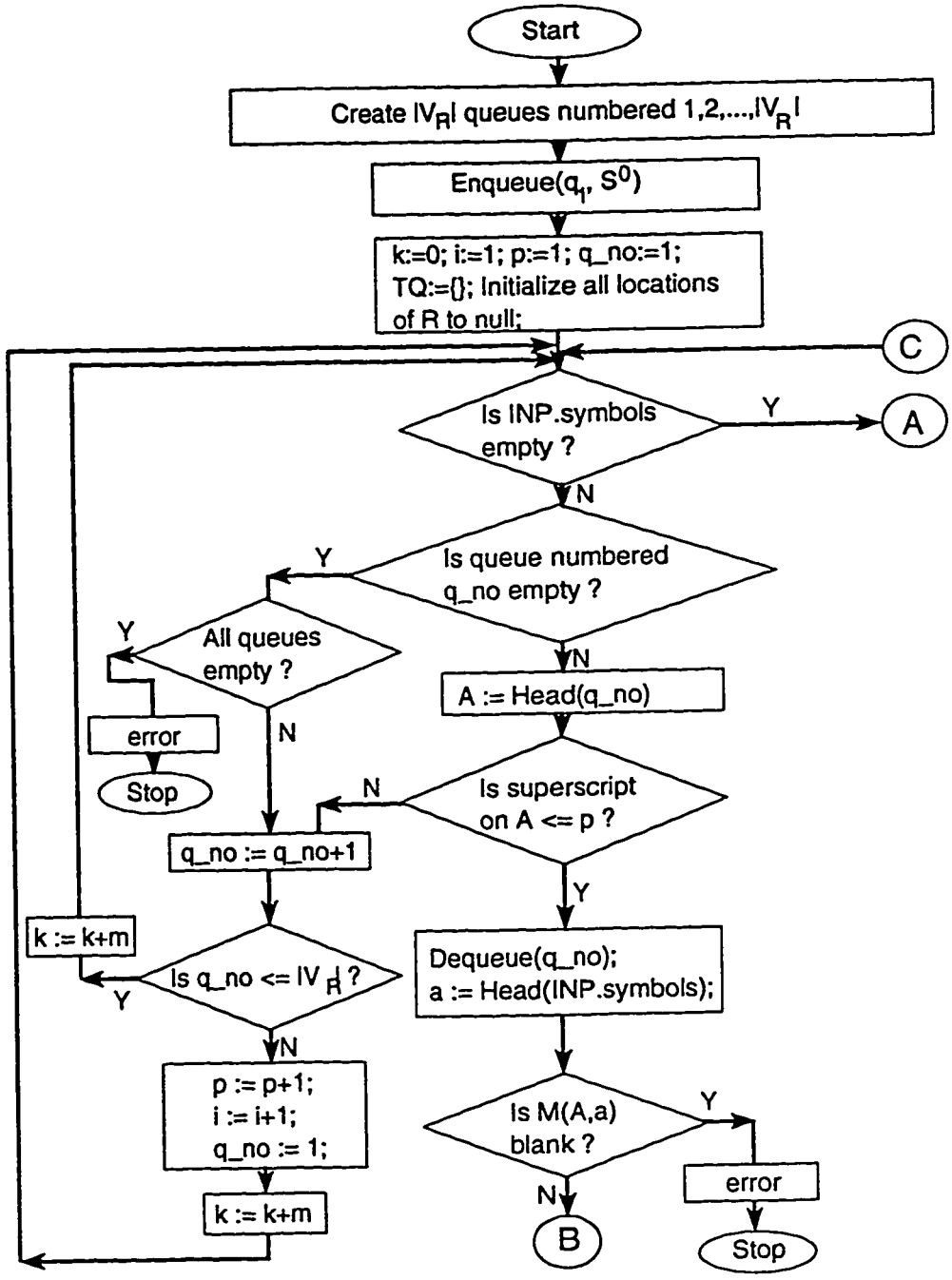

Figure 5.7: Flowchart for MRG/1 Predictive Parser.

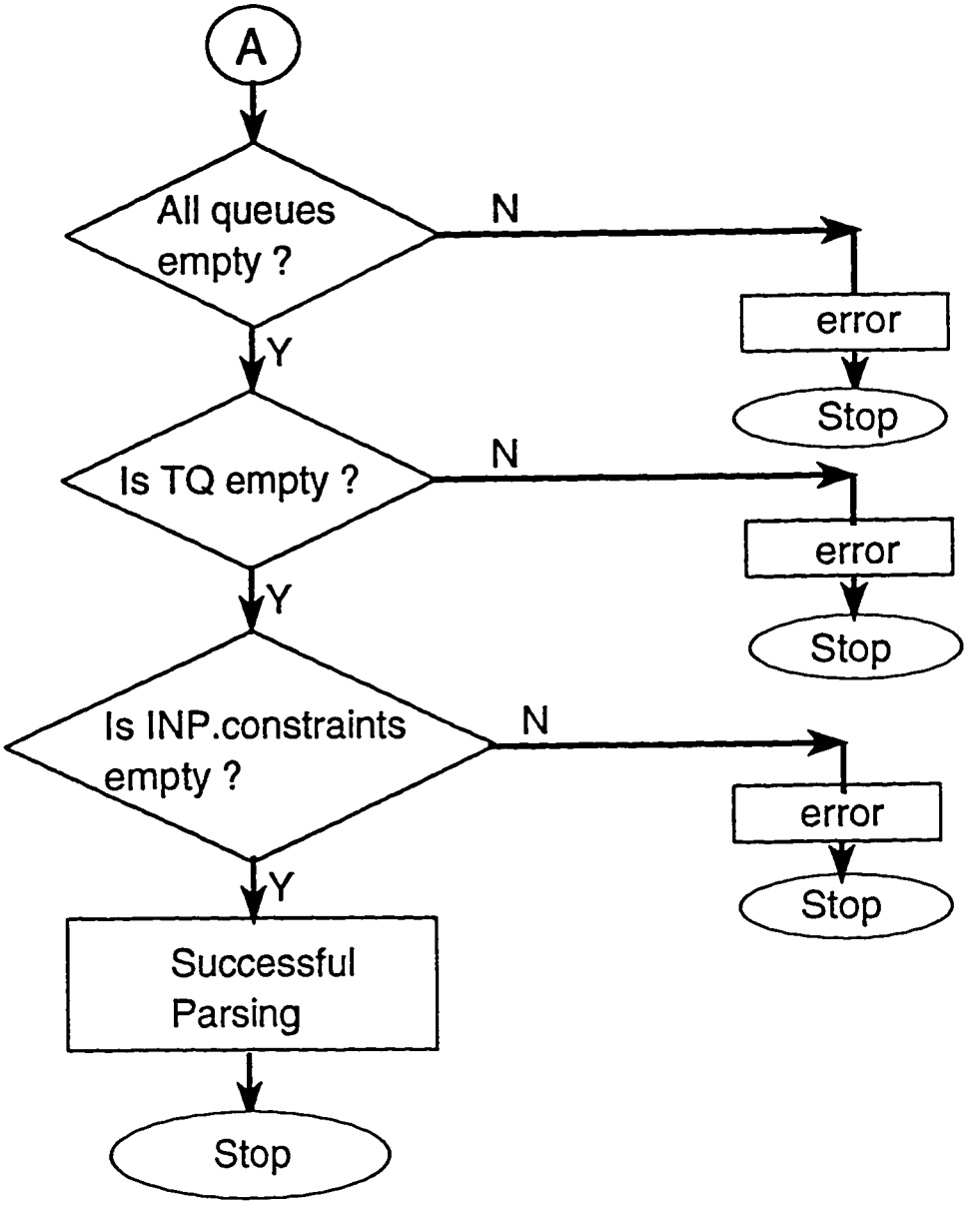

Figure 5.8: Flowchart for MRG/1 Predictive Parser contd...

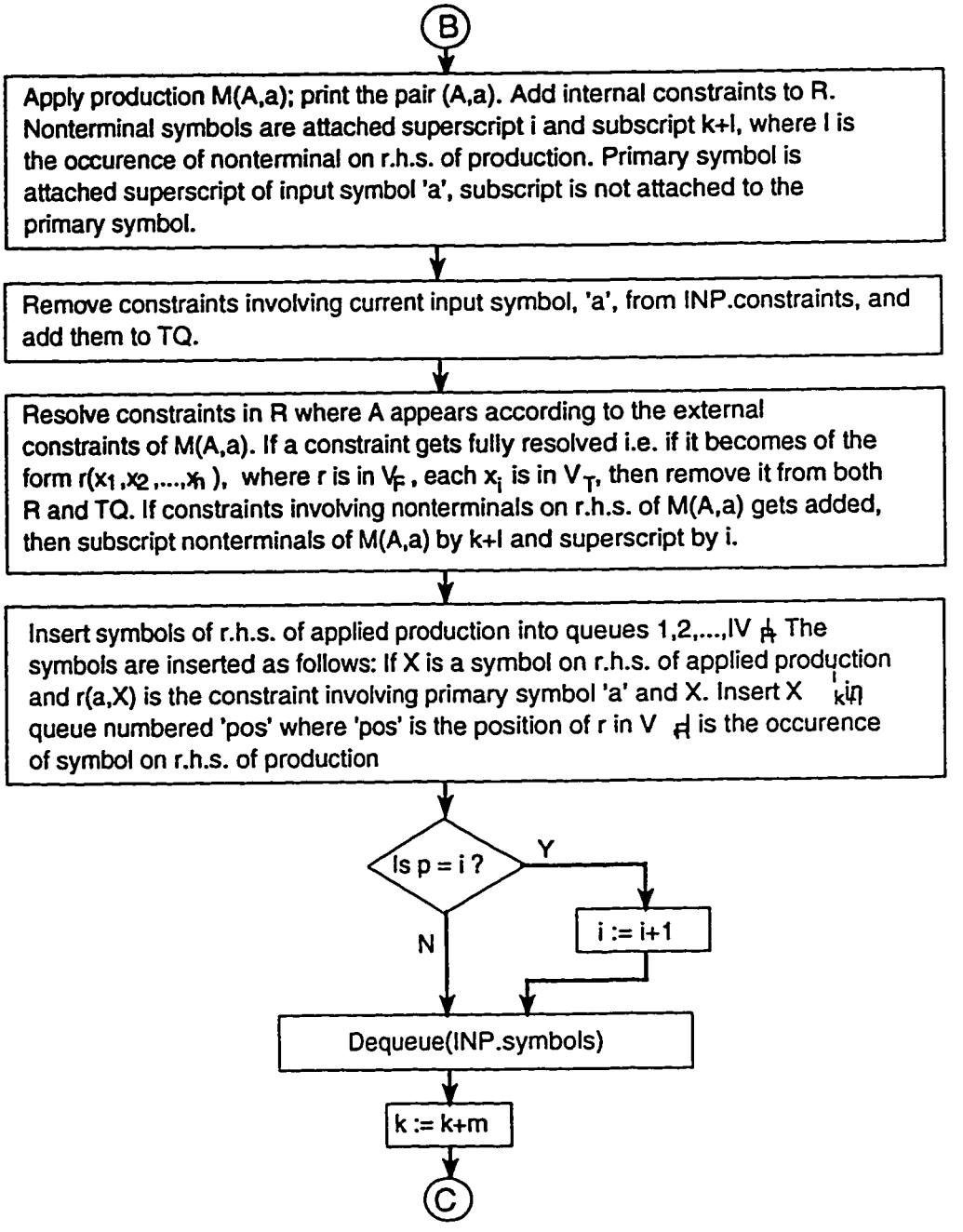

Figure 5.9: Flowchart for MRG/1 Predictive Parser contd...

The algorithmic description is self explanatory. However to clarify some of the issues, let us discuss about some of the operations performed.

Initially $|V_R|$ queues numbered 1,2,...,$|V_R|$ are created. The purpose of these queues is to help in reordering of the intermediate sentential forms. The intermediate sentential forms are reordered according to the ordering of spatial relations. The purpose of this reordering is to ensure that all the nonterminals in the parse tree at a level i are expanded before any nonterminal at level i+1 is expanded. In other words, the reordering ensures that the parse tree is traversed (produced) in a breadth first manner. The reordering is achieved by the use of $|V_R|$ queues and superscripts on the nonterminals.

Superscripts are attached to nonterminals before they are inserted into the queues. Also superscripts are attached to nonterminals in the internal constraints, before these constraints are added to R. These superscripts denotes the level of nonterminals in the parse tree. The superscript attached to a nonterminal during parsing is not the same as the superscript on it in the r.h.s. of applied production. The superscript attached to a nonterminal during parsing is in no way related to the superscript attached to it in the r.h.s. of a production. The superscript on a nonterminal in r.h.s. of a production denote its occurence on the r.h.s. of that production. Whereas the sole purpose of the superscript attached to a nonterminal during parsing is to indicate its level in the parse tree so that the parse tree is traversed in a top down breadth first manner. In other words, superscripts attached to nonterminals ensures that all the nonterminals at level i are expanded before expanding any nonterminal at level i+1.

Queues are used to expand nonterminals at a particular level from left to right. This

process becomes clear when we apply our parsing algorithm to examples in the next section. Each nonterminal symbol is subscripted also, prior to its insertion into the queues. Also when the constraints are added to R, the nonterminals in the constraints are both super and subscripted. Subscripts are used to distinguish between different occurences of the same nonterminal, at the same level, in the parse tree. This distinction is necessary in order to resolve the constraints in R. The subscript attached to a nonterminal will have a value k+l, where l is the occurence of nonterminal on r.h.s. of current production, and k is a counter used for assigning unique subscripts to symbols.

The value of k is incremented by m at the end of each iteration, where m is the maximum number of symbols excluding the primary symbol, on r.h.s. of any production in the production set. The reason for incrementing the value of k by m is the following.

Let us say that a production, having m different occurences of a nonterminal on its r.h.s., is applied at a parsing step. The production would look like

$A := \{a, Y^1, Y^2, ..., Y^m\}$ {internal constraints} {external constraints}

Assuming k has a value 0, the different occurences of symbol Y on r.h.s. will be assigned subscripts 1,2,...,m, before being inserted into queues and as part of constraints in R. When the same production is applied again at the next iteration, the different occurences of Y should have different subscripts (different from the subsripts of already inserted Ys in the queues). For achieving this we have to increment the value of k by m, which is the number of symbols on the r.h.s. of production excluding the primary symbol.

The parsing algorithm is applied to few example sentences and steps are traced in the next section. This helps in further clarifying the working of the algorithm.

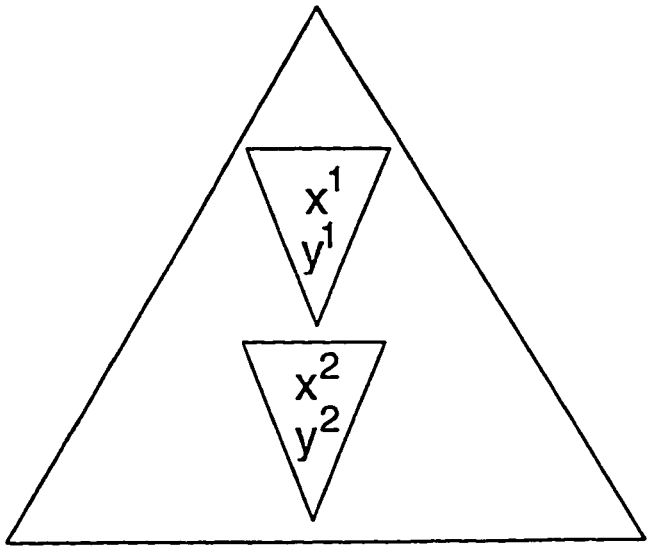

Figure 5.10: Sample iconic sentence reproduced from Figure 4.1.

5.3 Examples

Example 5.1: Let us apply our parsing algorithm to the iconic sentence of example 4.4. The sentence is reproduced in Figure 5.10 for sake of convenience. The MRG/1 formalism for this sentence is given in example 4.4. It is also reproduced below for sake of convenience.

G := (V_N,V_T,V_R,S,P) := ({S,A,B},{$\triangle$,$\triangledown$,x,y},{enclose,above},S,P) where P consists of

1. S := {Δ,A^1,A^2} {enclose(Δ,A^1),enclose(Δ,A^2),above(A^1,A^2)}

2. A := {∇,B^1,B^2} {enclose(∇,B^1),enclose(∇,B^2),above(B^1,B^2)} {enclose(Δ^m,A) :- enclose(Δ^m,∇); above(A^m,A) :- above(A^m,∇); above(A,A^m) :- above(∇,A^m); above(∇^m,A) :- above(∇^m,∇); above(A,∇^m) :- above(∇,∇^m);}

3. B := {x} {} {enclose(∇^m,B) :- enclose(∇^m,x); above(B^m,B) :- above(B^m,x); above(B,B^m) :- above(x,B^m)}

4. B := {y} {} {enclose(∇^m,B) :- enclose(∇^m,y); above(B^m,B) :- above(B^m,y); above(B,B^m) :- above(y,B^m); above(x^m,B) :- above(x^m,y)}

The sentence in Figure 5.10 can be ordered and represented as w = (T,RT) = ({Δ^1,∇^1,∇^2,x^1,y^1,x^2,y^2},{enclose(Δ^1,∇^1),enclose(Δ^1,∇^2),enclose(∇^1,x^1), enclose(∇^1,y^1),above(∇^1,∇^2),enclose(∇^2,x^2),enclose(∇^2,y^2),above(x^1,y^1), above(x^2,y^2)}).

Steps involve in parsing this sentence are as follows:

Initially: INP := ({Δ^1,∇^1,∇^2,x^1,y^1,x^2,y^2},{enclose(Δ^1,∇^1),enclose(Δ^1,∇^2), enclose(∇^1,x^1),enclose(∇^1,y^1),above(∇^1,∇^2),enclose(∇^2,x^2),enclose(∇^2,y^2), above(x^1,y^1),above(x^2,y^2)})
q_1:={S^0}; q_2:={}; i:=1; p:=1; q_no:=1; k:=0; TQ:={}; R:={};

Iteration 1: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing queue q_no, and assigning Head(INP.symbols) to a, we get
A:=S^0; q_1:={}; q_2:={}; a:=Δ^1;

M(A,a) is not blank, so Production 1 is applied. After adding internal constraints to R we get

R:= {enclose(Δ^1,A_1^1),enclose(Δ^1,A_2^1),above(A_1^1,A_2^1)}

There are no constraints in R involving S^0, so no constraints are resolved.

After removing constraints involving a from INP.constraints and adding to TQ we get

TQ:={enclose(Δ^1,∇^1),enclose(Δ^1,∇^2)}

There are no constraints common to both R and TQ, so nothing is removed from R and TQ.

After inserting symbols into the queues we get

q_1:={A_1^1,A_2^1}; q_2:={}; i:=2;

After dequeuing INP.symbols, we get

INP := ({∇^1,∇^2,x^1,y^1,x^2,y^2},{enclose(∇^1,x^1),enclose(∇^1,y^1),above(∇^1,∇^2),
enclose(∇^2,x^2),enclose(∇^2,y^2),above(x^1,y^1),above(x^2,y^2)})

i:=2; k:=2;

Iteration 2: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing queue q_no, and assigning Head(INP.symbols) to a, we get

A:=A_1^1; q_1:={A_2^1}; q_2:={}; a:=∇^1;

M(A,a) is not blank, so Production 2 is applied. After adding internal constraints to R we get

R:= {enclose(Δ^1,A_1^1),enclose(Δ^1,A_2^1),above(A_1^1,A_2^1),enclose(∇^1,B_3^2),
enclose(∇^1,B_4^2),above(B_3^2,B_4^2)}

After resolving constraints in R we get

{enclose(Δ^1,∇^1),enclose(Δ^1,A_2^1),above(∇^1,A_2^1),enclose(∇^1,B_3^2),enclose(∇^1,B_4^2),
above(B_3^2,B_4^2)}

After removing constraints constraints involving a from INP.constraints and adding to TQ we get

TQ:={{enclose(Δ^1,∇^1),enclose(Δ^1,∇^2),enclose(∇^1,x^1),enclose(∇^1,y^1),above(∇^1,∇^2)};

The constraint enclose(Δ^1,∇^1) is common to both R and TQ, so it is removed from both R and TQ. R and TQ becomes

R:={enclose(Δ^1,A_2^1),above(∇^1,A_2^1),enclose(∇^1,B_3^2),enclose(∇^1,B_4^2),above(B_3^2,B_4^2)}

TQ:={enclose(Δ^1,∇^2),enclose(∇^1,x^1),enclose(∇^1,y^1),above(∇^1,∇^2)}

After inserting symbols into queues we get

q_1:={A_2^1,B_3^2,B_4^2}; q_2:={};

After dequeuing INP.symbols we get

INP := ({∇^2,x^1,y^1,x^2,y^2},{enclose(∇^2,x^2),enclose(∇^2,y^2),above(x^1,y^1),above(x^2,y^2)})

K:=4;

Iteration 3: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get

A:=A_2^1; q_1:={B_3^2,B_4^2}; q_2:={}; a:=∇^2;

M(A,a) is not blank, so Production 2 is applied. After adding internal constraints to R we get

R:={enclose(Δ^1,A_2^1),above(∇^1,A_2^1),enclose(∇^1,B_3^2),enclose(∇^1,B_4^2),above(B_3^2,B_4^2),
enclose(∇^2,B_5^2),enclose(∇^2,B_6^2),above(B_5^2,B_6^2)}

After resolving constraints in R we get

R:={enclose(Δ^1,∇^2),above(∇^1,∇^2),enclose(∇^1,B_3^2),enclose(∇^1,B_4^2),above(B_3^2,B_4^2),

enclose(∇^2,B_5^2),enclose(∇^2,B_6^2),above(B_5^2,B_6^2)}

After removing constraints involving a from INP.constraints and adding to TQ, we get

TQ:={enclose(Δ^1,∇^2),enclose(∇^1,x^1),enclose(∇^1,y^1),above(∇^1,∇^2),enclose(∇^2,x^2), enclose(∇^2,y^2)}

The constraints enclose(Δ^1,∇^2),above(∇^1,∇^2) are removed from both R and TQ, so R and TQ becomes

R:={enclose(∇^1,B_3^2),enclose(∇^1,B_4^2),above(B_3^2,B_4^2),enclose(∇^2,B_5^2),enclose(∇^2,B_6^2), above(B_5^2,B_6^2)}

TQ:={enclose(∇^1,x^1),enclose(∇^1,y^1),enclose(∇^2,x^2),enclose(∇^2,y^2)}

After inserting symbols into queues we get

q_1:={B_3^2,B_4^2,B_5^2,B_6^2}; q_2:={};

After dequeuing INP.symbols we get

INP := ({x^1,y^1,x^2,y^2},{above(x^1,y^1),above(x^2,y^2)})

K:=6;

Iteration 4: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, we see that the superscript on A is not less than or equal to p. So we increment the q_no, which now becomes 2; k=8; and we go on to the next iteration.

Iteration 5: In this iteration we find queue numbered 2 to be empty. We increment q_no, which now exceeds $|V_R|$. So q_no:=1; p:=2; i:=3; k:=10. We now go to the next iteration.

Iteration 6: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get

$A:=B_3^2$; $q_1:=\{B_4^2,B_5^2,B_6^2\}$; $q_2:=\{\}$; $a:=x^1$;

M(A,a) is not blank, so Production 3 is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving constraints R becomes

$R:=\{enclose(\nabla^1,x^1),enclose(\nabla^1,B_4^2),above(x^1,B_4^2),enclose(\nabla^2,B_5^2),enclose(\nabla^2,B_6^2),$
$above(B_5^2,B_6^2)\}$

After removing constraints involving a from INP.constraints and adding to TQ we get

$TQ:=\{enclose(\nabla^1,x^1),enclose(\nabla^1,y^1),enclose(\nabla^2,x^2),enclose(\nabla^2,y^2),above(x^1,y^1)\}$

After removing the constraint $enclose(\nabla^1,x^1)$ from R and TQ we get

$R:=\{enclose(\nabla^1,B_4^2),above(x^1,B_4^2),enclose(\nabla^2,B_5^2),enclose(\nabla^2,B_6^2),above(B_5^2,B_6^2)\}$

$TQ:=\{enclose(\nabla^1,y^1),enclose(\nabla^2,x^2),enclose(\nabla^2,y^2),above(x^1,y^1)\}$

There are no symbols in this production to be inserted into the queues.

After dequeuing INP.symbols we get

$INP := (\{y^1,x^2,y^2\},\{above(x^2,y^2)\})$

k:=12;

Iteration 7: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get

$A:=B_4^2$; $q_1:=\{B_5^2,B_6^2\}$; $q_2:=\{\}$; $a:=y^1$;

M(A,a) is not blank, so Production 4 is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving constraints R becomes

R:={enclose(∇^1,y^1),above(x^1,y^1),enclose(∇^2,B_5^2),enclose(∇^2,B_6^2),above(B_5^2,B_6^2)}

Nothing is added to TQ at this step, so TQ remains

TQ:={enclose(∇^1,y^1),enclose(∇^2,x^2),enclose(∇^2,y^2),above(x^1,y^1)}

After removing the constraints enclose(∇^1,y^1), above(x^1,y^1) from R and TQ we get

R:={enclose(∇^2,B_5^2),enclose(∇^2,B_6^2),above(B_5^2,B_6^2)}

TQ:={enclose(∇^2,x^2),enclose(∇^2,y^2)}

There are no symbols in this production to be inserted into the queues.

After dequeuing INP.symbols we get

INP := ({x^2,y^2},{above(x^2,y^2)})

k:=14;

Iteration 8: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get

A:=B_5^2; q_1:={B_6^2}; q_2:={}; a:=x^2;

M(A,a) is not blank, so Production 3 is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving constraints R becomes

R:={enclose(∇^2,x^2),enclose(∇^2,B_6^2),above(x^2,B_6^2)}

Removing the constraint involving x^2 from INP.constraints and adding it to TQ we get

TQ:={enclose(∇^2,x^2),enclose(∇^2,y^2),above(x^2,y^2)}

After removing the constraint enclose(∇^2,x^2) from R and TQ we get

R:={enclose(∇^2,B_6^2),above(x^2,B_6^2)}

TQ:={enclose(∇^2,y^2),above(x^2,y^2)}

There are no symbols in this production to be inserted into the queues.

After dequeuing INP.symbols we get

INP := ({y^2},{})

k:=16;

Iteration 9: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get

A:=B_6^2; q_1:={}; q_2:={}; a:=y^2;

M(A,a) is not blank, so Production 4 is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving constraints R becomes

R:={enclose(∇^2,y^2),above(x^2,y^2)}

Since INP.constraints is empty,there are no constraints remaining to be added to TQ; After removing the constraints enclose(∇^2,y^2),above(x^2,y^2) from R and TQ we get

R:={}

TQ:={}

There are no symbols in this production to be inserted into the queues.

After dequeuing INP.symbols we get

INP := ({},{})

k:=18;

Iteration 10: In this iteration INP.symbols is empty. Checks are made to see

whether all queues are empty or not, R is empty or not, TQ is empty or not, and INP.constraints is empty or not. All these checks succeed; at this time the algorithm announces successful completion of parsing.

Example 5.2: Let us now try to parse a sentence of the language described in example 4.3. Sentences in this language consists of 2D grids of two kinds of trees. Trees of type ♠ are placed on the South-East frontier, remaining grid positions are filled by ♣ trees. A sample sentence of this language is v shown next:

$$v = \begin{matrix} \clubsuit^1 & \clubsuit^2 & \spadesuit^2 \\ \spadesuit^1 & \spadesuit^3 & \spadesuit^4 \end{matrix}$$

The MRG/1 for this language is given in example 4.3. It is reproduced here for the sake of convenience. It is written as G := (V_N,V_T,V_R,S,P) := ({S,B,A},{♣,♠},{x,y,z},S,P) where P consists of

1. S := {♣,B^1,A^1} {x(♣,B^1),y(♣,A^1),z(B^1,A^1)}
2. B := {♣,B^1,A^1} {x(♣,B^1),y(♣,A^1),z(B^1,A^1)} {x($♣^m$,B) :- x($♣^m$,♣); z(B,A^m) :- z(♣,A^m),x(A^m,A^1)}
3. B := {♠,A^1} {y(♠,A^1)} {x($♣^m$,B) :- x($♣^m$,♠); z(B,A^m) :- z(♠,A^m),x(A^m,A^1)}
4. A := {♣,A^1} {y(♣,A^1)} {y($♣^m$,A) :- y($♣^m$,♣); z(A,A^m) :- z(♣,A^m),x(A^m,A^1); z($♠^m$,A) :- z($♠^m$,♣); z($♣^m$,A) :- z($♣^m$,♣); x(A,A^m) :- x(♣,A^m),z(A^m,A^1); x($♣^m$,A) :- x($♣^m$,♣)}

5. A := {♠} {} {y(♠^{m},A) :- y(♠^{m},♠); y(♣^{m},A) :- y(♣^{m},♠); z(♠^{m},A) :- z(♠^{m},♠); z(♣^{m},A) :- z(♣^{m},♠); z(A,A^m) :- z(♠,A^m); x(A,A^m) :- x(♠,A^m); x(♠^{m},A) :- x(♠^{m},♠); x(A^m,A) :- x(A^m,♠)}

Assuming the ordering of relations to be x<y<z, the sentence v can be represented as v := (T,RT) := ({♣^{1},♣^{2},♠^{1},♠^{2},♠^{3},♠^{4}}{x(♣^{1},♣^{2}),y(♣^{1},♠^{1}),x(♣^{2},♠^{2}), y(♣^{2},♠^{3}),z(♣^{2},♠^{1}),x(♠^{1},♠^{3}),y(♠^{2},♠^{4}),z(♠^{2},♠^{3}),x(♠^{3},♠^{4})}.

The steps involved in parsing this sentence are as follows.

Initially: INP := ({♣^{1},♣^{2},♠^{1},♠^{2},♠^{3},♠^{4}}{x(♣^{1},♣^{2}),y(♣^{1},♠^{1}),x(♣^{2},♠^{2}),y(♣^{2},♠^{3}), z(♣^{2},♠^{1}),x(♠^{1},♠^{3}),y(♠^{2},♠^{4}),z(♠^{2},♠^{3}),x(♠^{3},♠^{4})}.
q_1:={S^0}; q_2:={}; q_3:={}; i:=1; p:=1; q_no:=1; k:=0; TQ:={}; R:={};

Iteration 1: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get
A:=S^0; q_1:={}; q_2:={}; q_3:={}; a:=♣^{1};
M(A,a) is not blank, so Production 1 is applied. After adding internal constraints to R we get
R:= {x(♣^{1},B_1^1),y(♣^{1},A_1^1),z(B_1^1,A_1^1)}
There are no constraints in R involving S^0, so no constraints are resolved.
After removing constraints involving a from INP.constraints and adding to TQ we get
TQ:={x(♣^{1},♣^{2}),y(♣^{1},♠^{1})}
There are no constraints common to both R and TQ, so nothing is removed from R

and TQ.

After inserting symbols into the queues we get

q_1:={B_1^1}; q_2:={A_1^1}; q_3:={}; i:=2;

After dequeuing INP.symbols, we get

INP := ({$\clubsuit^2$,$\spadesuit^1$,$\spadesuit^2$,$\spadesuit^3$,$\spadesuit^4$}{x($\clubsuit^2$,$\spadesuit^2$),y($\clubsuit^2$,$\spadesuit^3$),z($\clubsuit^2$,$\spadesuit^1$),x($\spadesuit^1$,$\spadesuit^3$),y($\spadesuit^2$,$\spadesuit^4$),

z($\spadesuit^2$,$\spadesuit^3$),x($\spadesuit^3$,$\spadesuit^4$)}.

i:=2; k:=2;

Iteration 2: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get

A:=B_1^1; q_1:={}; q_2:={A_1^1}; q_3:={}; a:=$\clubsuit^2$;

M(A,a) is not blank, so Production 2 is applied. After adding internal constraints to R we get

R:= {x($\clubsuit^1$,B_1^1),y($\clubsuit^1$,A_1^1),z(B_1^1,A_1^1),x($\clubsuit^2$,B_3^2),y($\clubsuit^2$,A_3^2),z(B_3^2,A_3^2)}

After resolving constraints in R we get

R:={x($\clubsuit^1$,$\clubsuit^2$),y($\clubsuit^1$,A_1^1),z($\clubsuit^2$,A_1^1),x(A_1^1,A_3^2),x($\clubsuit^2$,B_3^2),y($\clubsuit^2$,A_3^2),z(B_3^2,A_3^2)}

After removing constraints constraints involving a from INP.constraints and adding to TQ we get

TQ:={x($\clubsuit^1$,$\clubsuit^2$),y($\clubsuit^1$,$\spadesuit^1$),x($\clubsuit^2$,$\spadesuit^2$),y($\clubsuit^2$,$\spadesuit^3$),z($\clubsuit^2$,$\spadesuit^1$)}

The constraint x($\clubsuit^1$,$\clubsuit^2$) is common to both R and TQ, so it is removed from both R and TQ. R and TQ becomes

R:={y($\clubsuit^1$,A_1^1),z($\clubsuit^2$,A_1^1),x(A_1^1,A_3^2),x($\clubsuit^2$,B_3^2),y($\clubsuit^2$,A_3^2),z(B_3^2,A_3^2)}

TQ:={y($\clubsuit^1$,$\spadesuit^1$),x($\clubsuit^2$,$\spadesuit^2$),y($\clubsuit^2$,$\spadesuit^3$),z($\clubsuit^2$,$\spadesuit^1$)}

After inserting symbols into queues we get

q_1:={B_3^2}; q_2:={A_1^1,A_3^2}; q_3:={};

After dequeuing INP.symbols we get

INP := ({$\spadesuit^1$,$\spadesuit^2$,$\spadesuit^3$,$\spadesuit^4$}{x($\spadesuit^1$,$\spadesuit^3$),y($\spadesuit^2$,$\spadesuit^4$),z($\spadesuit^2$,$\spadesuit^3$),x($\spadesuit^3$,$\spadesuit^4$)}.

K:=4;

Iteration 3: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, we see that the superscript on A is not less than or equal to p. So we increment the q_no, which now becomes 2; k=6; and we go on to the next iteration.

Iteration 4: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get

A:=A_1^1; q_1:={B_3^2}; q_2:={A_3^2}; a:=$\spadesuit^1$;

M(A,a) is not blank, so Production 5 is applied. Nothing is added to R since there are no internal constraints in this production.

After resolving constraints R becomes

R:={y($\clubsuit^1$,$\spadesuit^1$),z($\clubsuit^2$,$\spadesuit^1$),x($\spadesuit^1$,A_3^2),x($\clubsuit^2$,B_3^2),y($\clubsuit^2$,A_3^2),z(B_3^2,A_3^2)}

After adding constraints involving a from INP.constraints and adding it to TQ we get

TQ:={y($\clubsuit^1$,$\spadesuit^1$),x($\clubsuit^2$,$\spadesuit^2$),y($\clubsuit^2$,$\spadesuit^3$),z($\clubsuit^2$,$\spadesuit^1$),x($\spadesuit^1$,$\spadesuit^3$)}

After removing the constraints y($\clubsuit^1$,$\spadesuit^1$),z($\clubsuit^2$,$\spadesuit^1$) from R and TQ we get

R:={x($\spadesuit^1$,A_3^2),x($\clubsuit^2$,B_3^2),y($\clubsuit^2$,A_3^2),z(B_3^2,A_3^2)}

TQ:={x($\clubsuit^2$,$\spadesuit^2$),y($\clubsuit^2$,$\spadesuit^3$),x($\spadesuit^1$,$\spadesuit^3$)}

There are no symbols in this production to be inserted into the queues.

After dequeuing INP.symbols we get

INP := ({$\spadesuit^2$,$\spadesuit^3$,$\spadesuit^4$}{y($\spadesuit^2$,$\spadesuit^4$),z($\spadesuit^2$,$\spadesuit^3$),x($\spadesuit^3$,$\spadesuit^4$)}.

k:=8;

Iteration 5: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, we see that the superscript on A is not less than or equal to p. So we increment the q_no which now becomes 3; k=10; and we go on to the next iteration.

Iteration 6: In this iteration we find queue numbered 3 to be empty. We increment q_no, which now exceeds $|V_R|$. So q_no:=1; p:=2; i:=3; k:=12. We now go to the next iteration.

Iteration 7: After checking that the queue q_no is not empty, assigning the Head(q_no) to A, dequeuing it, and assigning Head(INP.symbols) to a, we get

A:=B_3^2; q_1:={}; q_2:={A_3^2}; a:=$\spadesuit^2$;

M(A,a) is not blank, so Production 3 is applied. After adding internal constraints to R we get.

R:={x($\spadesuit^1$,A_3^2),x($\clubsuit^2$,B_3^2),y($\clubsuit^2$,A_3^2),z(B_3^2,A_3^2),y($\spadesuit^2$,A_{13}^3)}

After resolving constraints R becomes

R:={x($\spadesuit^1$,A_3^2),x($\clubsuit^2$,$\spadesuit^2$),y($\clubsuit^2$,A_3^2),z($\spadesuit^2$,A_3^2),x(A_3^2,A_{13}^3),y($\spadesuit^2$,A_{13}^3)}

After removing constraints involving a from INP.constraints and adding it to TQ we get

TQ:={x($\clubsuit^2$,$\spadesuit^2$),y($\clubsuit^2$,$\spadesuit^3$),x($\spadesuit^1$,$\spadesuit^3$),y($\spadesuit^2$,$\spadesuit^4$),z($\spadesuit^2$,$\spadesuit^3$)}

After removing the constraint x($\clubsuit^2$,$\spadesuit^2$) from R and TQ we get

R:={x($\spadesuit^1$,A_3^2),y($\clubsuit^2$,A_3^2),z($\spadesuit^2$,A_3^2),x(A_3^2,A_{13}^3),y($\spadesuit^2$,A_{13}^3)}

TQ:={y(♣^{2},♠^{3}),x(♠^{1},♠^{3}),y(♠^{2},♠^{4}),z(♠^{2},♠^{3})}

After inserting symbols into queues we get

q_1:={}; q_2:={A^2_3,A^3_{13}}; q_3:={};

After dequeuing INP.symbols we get

INP := ({♠^{3},♠^{4}}{x(♠^{3},♠^{4})}.

k:=14;

Iteration 8: In this iteration we find queue numbered 1 to be empty. We increment q_no, which now becomes 2; k:=16. We now go to the next iteration.

Iteration 9: A:=A^2_3; q_1:={}; q_2:={A^3_{13}}; q_3:={}; a:=♠^{3};

Production 5 is applied. Nothing is added to R, since there are no internal constraints.

After resolving constraints in R we get

R:={x(♠^{1},♠^{3}),y(♣^{2},♠^{3}),z(♠^{2},♠^{3}),x(♠^{3},A^3_{13}),y(♠^{2},A^3_{13})}

After removing constraints involving a from INP.constraints and adding to TQ we get

TQ:={y(♣^{2},♠^{3}),x(♠^{1},♠^{3}),y(♠^{2},♠^{4}),z(♠^{2},♠^{3}),x(♠^{3},♠^{4})}

The constraints x(♠^{1},♠^{3}),y(♣^{2},♠^{3}),z(♠^{2},♠^{3}) are removed from R and TQ, and we get

R:={x(♠^{3},A^3_{13}),y(♠^{2},A^3_{13})}

TQ:={y(♠^{2},♠^{4}),x(♠^{3},♠^{4})}

Nothing is inserted into the queues. After dequeuing INP.symbols we get

INP := ({♠^{4}}{}).

k:=18;

Iteration 10: A:=A^3_{13}; since superscript on A is not less than or equal to p, we go to queue 3. So q_no:=3, k:=20.

Iteration 11: Since queue 3 is empty, we go back to queue 1 with: p:=3, i:=4, q_no:=1, k:=22.

Iteration 12: Since queue 1 is empty, we go to queue 2. So q_no:=2, k:=24.

Iteration 13: A:=A^3_{13}; q_1:={}; q_2:={}; q_3:={}; a:=$\spadesuit^4$;

Production 5 is applied. Nothing is added to R. After resolving constraints in R we get

R:={x($\spadesuit^3$,$\spadesuit^4$),y($\spadesuit^2$,$\spadesuit^4$)}

Since there are no constraints in INP.constraints nothing is added to TQ. Removing the constraints x($\spadesuit^3$,$\spadesuit^4$),y($\spadesuit^2$,$\spadesuit^4$) from R and TQ we get

R:={}; TQ:={};

Nothing is inserted into the queues. After dequeuing INP.symbols, INP becomes

INP := ({}{}).

k:=26;

Iteration 14: In this iteration INP.symbols is empty. Checks are made to see whether all queues are empty or not, R is empty or not, TQ is empty or not, and INP.constraints is empty or not. All these checks succeed; at this time the algorithm

announces successful completion of parsing.

5.4 Time Complexity

Lemma 5.1: The number of times the main while loop of the parsing algorithm is repeated is n*$|V_R|$ or O(n).

Proof: In the worst case symbols are added to only one queue, remaining queues are kept empty. Also only one symbol is added to the queue at a time. Let us assume that after application of first production (first iteration of while loop or first step of parsing), symbols are added only to the last queue (i.e. queue numbered $|V_R|$). The initial configuration of the queues will be as follows:

q_1:={S^0}; q_2:={}; q_3:={};,,,,; $q_{|V_R|}$:={}.

In the first iteration S^0 in queue 1 will be expanded and a nonterminal symbol, let us say A (written as A_1^1 with superscript and subscript), will be inserted into queue numbered $|V_R|$. In the second iteration queue numbered 1 is found to be empty so the parser goes to queue 2. Since queue 2 is also empty, it goes to queue 3, and so on until it reaches queue $|V_R|$. The number of iterations, after the first iteration, required to reach queue $|V_R|$ is $|V_R|$-1. Once queue $|V_R|$ is reached, the symbol A_1^1 will be expanded to another symbol, let us say B (written as B_s^2 where s=$|V_R|$*m+1). By the end of this iteration only two symbols of the input sentence are parsed. The first one is parsed when S is expanded, and the second one when A is expanded. The parsed symbols are primary symbols of applied productions. Still (n-2) symbols of the input sentence are to be parsed.

In the next iteration (the iteration immediately after the iteration in which A is expanded) the parser finds the superscript of B to be greater than p (remember that p

is initialized to 1), so it loops back and goes to q_1 after incrementing i and p. It finds q_1 to be empty, goes to q_2, and so on till it reaches $q_{|V_R|}$. At this point it expands B. So to expand every nonterminal on $q_{|V_R|}$ the parser performs $|V_R|+1$ iterations. Since there are (n-2) symbols yet to be analyzed, the parser needs $(n-2)*(|V_R|+1)$ iterations. Therefore the total iterations required to parse a sentence of 'n' symbols

=

1 iteration for expansion of S + $(|V_R|-1)$ iterations required to reach queue $|V_R| + 1$ iteration to expand A + $(n-2)*(|V_R|+1)$ iterations required to parse remaining n-2 symbols

$= 1+|V_R|-1+1+(n-2)*(|V_R|+1) = (n-1)*(|V_R|+1)$.

Assuming $n>>1$, $|V_R|>>1$ we get

$n*|V_R|$

So in the worst case the while loop is repeated $n*|V_R|$ times. Since $|V_R|$ is constant for a given grammar, we can say that the number of times the main while loop is repeated is O(n).

Lemma 5.2: The time complexity of the parsing algorithm is O(n).

Proof: Let us derive the time complexity for the parsing algorithm. The statement for creating $|V_R|$ queues takes $O(|V_R|)$ time. The statements for initializing queue 1; for initializing variables k,i,p,q_no; and initializing TQ and R takes constant time. The main while loop of the algorithm is repeated at most $n*|V_R|$ times(see lemma 5.1). Within the while loop, the time taken by various statements is as follows:

The statement which checks whether INP.symbols is empty, is executed in constant time. The statement which checks whether all queues are empty takes $O(|V_R|)$ time.

The following blocks of statements also takes constant time.

- Is queue numbered q_no empty
- A:=Head(q_no)
- Is superscript on A≤p
- Dequeue(q_no)
  a:=Head(INP.symbols)
- q_no:=q_no+1
- Is q_no=$|V_R|$+1
- p:=P+1
  i:i+1
  q_no:=1
- Is TQ empty
- Is INP.constraints empty

Since the parse table is organized as 2D array indexed by nonterminals along the rows and terminals along the columns, the statement 'Is M(A,a) blank ?' takes constant time. The time taken for applying a production M(A,a) and adding its internal constraints to R is constant; because once the grammar is given we can consider number of symbols and internal constraints of any production as constants. Since the constraints of input sentence are totally ordered, the constraints involving current input symbol 'a' appears towards the begining of INP.constraints. The time taken to remove these constraints and add it to TQ, is constant. Since R is

of constant length and can be organized as an array, constraints involving current occurence of nonterminal A can be accessed in constant time. These constraints can then be resolved. A fully resolved constraint can be removed from both R and TQ. This operation also takes constant time. The time taken to insert symbols of r.h.s. of applied production (other than primary symbol) into queues is constant, because the number of symbols on r.h.s. of any MRG/1 production is considered to be constant, once the grammar is given. The time taken for the statements 'Is p=i' and 'Dequeue(INP.symbols)' is constant.

So time taken for 1 iteration of while loop is bounded by some constant, let us say δ_1. δ_1 could be $O(|V_R|)$, or δ, where δ is time taken by any other statement in the while loop. Since the while loop is repeated $n*|V_R|$ times; the time taken for execution of while loop = $n*|V_R|*\delta_1$.

Total time complexity of the algorithm = Time taken by execution of statements occuring before while loop + Time taken for execution of while loop. Since time taken by the execution of statements before the while loop is constant, time taken by the algorithm is determined by the time taken by the while loop.

So total time complexity of the algorithm = $n*|V_R|*\delta_1$. Since $|V_R|$ and δ_1 are constant, we get the asymptotic time complexity as O(n).

Lemma 5.3: When $|V_R| \rightarrow n$, i.e., when the size of relationship set approaches the size of input, the time complexity of MRG/1 based parsing algorithm becomes $O(n^2)$.

Proof: From lemma 5.1 we know that the maximum number of times the while

loop is performed is $n^*|V_R|$. From lemma 5.2 we know that the time taken for one iteration of the while loop is constant, let us say δ_1. So the time taken for all the iterations of while loop = $n^*|V_R|^*\delta_1$. When $|V_R| \rightarrow n$ we get time taken by while loop as $n^2*\delta_1$

Since δ_1 is constant, we say that time taken for while loop execution is $O(n^2)$. Since the time complexity of the while loop dominates (or determines) the time complexity of the parsing algorithm, we say that the time complexity of the parsing algorithm is $O(n^2)$.

5.5 Comparision of MRG/1 and RG/1 Based Parsing Algorithms

Since MRG/1 formalism is similar to RG/1 formalism, let us compare their parsing algorithms. Detailed RG/1 parsing algorithm with trace of an example is given in chapter 3 section 3.2. Detailed MRG/1 based parsing algorithm is given in section 5.2. Let us now compare these formalisms from the parsing efficiency point of view, and see the factors effecting them.

Parsing algorithms for both RG/1 and MRG/1 work top down and constructs the parse tree in a breadth first order. Both these algorithms set up loops which are repeated n times, where 'n' is the size of input sentence. The difference between these algorithms is in the way they resolve the internal constraints. In RG/1 based parsing algorithm, when the algorithm finds that two symbols are potentially con-

text identical it saves the information of all the dpci s (description of potentially contextual identities) in the set D. The time taken for constructing, inserting, and deleting dpcis is O(log n). Therefore we can say that the time taken for one iteration of for loop is O(log n). Since the loop is repeated n times, the time complexity of the loop, and hence the time complexity of the parsing algorithm is O(n log n).

In MRG/1 based parsing algorithm we do not use the concept of dpcis. We use super and subscripts on nonterminals to identify internal constraints involving them. The set of internal constraints which are not yet resolved can be organized as a 3D array indexed by nonterminals, superscripts, and subscripts. Since access time in an array is constant, the constraints involving a particular occurence of a nonterminal can be accessed in constant time. We have seen in lemma 5.2 that the time taken by all other statements in the while loop is constant. So we can say that the time taken for one iteration of the while loop is bounded by some constant. Since the loop is repeated n times, we can say that the time complexity of the loop, and hence the time complexity of the parsing algorithm is O(n).

When $|V_R| \rightarrow n$, the time complexity of the MRG/1 based parsing algorithm becomes $O(n^2)$ (see lemma 5.3). In RG/1 based parsing algorithm also, when $|V_R| \rightarrow n$ the time complexity becomes $O(n^2)$. This is because, when $|V_R| \rightarrow n$ the time taken to check whether two symbols are context identical or potentially context identical, within the for loop, becomes O(n). This time dominates the time taken for each iteration of the for loop. Since the for loop is repeated n times, the total time complexity of the parsing algorithm becomes $O(n^2)$.

When the number of symbols or number of internal constraints on r.h.s. of any MRG/1 production approaches n, the time complexity becomes $O(n^2)$. This is be-

cause, the time taken to apply a production, i.e., the time taken to add internal constraints to R, and the time taken to insert symbols into queues, becomes O(n). Since the loop is repeated n times, the complexity of the loop and consequently that of the parsing algorithm becomes $O(n^2)$. In this case in RG/1 based parsing algorithm also, the time taken to apply a production within the loop will be O(n). Since the loop is repeated n times the total time complexity of the loop and consequently that of the algorithm becomes $O(n^2)$.

Further the time taken to apply a production in RG/1 based parsing algorithm is more than the time taken to apply a production in MRG/1 parsing algorithm. This can be argued as follows:

In RG/1 based parsing algorithm, when we apply a production, we have to resolve all references to the nonterminal on l.h.s. of applied production, in the current sentential form. This requires time O(|R|*|p|), where |R| is the size of set of constraints of current sentential form, |p| is the size of set of evaluation rules. This complexity is derived by assuming that, for resolving each constraint we have to search the complete set of evaluation rules; and all the constraints in the current sentential form involves the nonterminal on l.h.s. of current production.

In MRG/1 based parsing algorithm, after applying the production M(A,a) and adding internal constraints, we resolve the constraints in R, where A appears (see the parsing algorithm). We resolve the constraints according to the external constraints of the current production M(A,a). This requires time O(|R|*|q|) where |R| is the size of R holding internal constraints, and |q| is the size of external constraints of current production. Since the evaluation rules are distributed as external constraints among various productions in MRG/1, $|q| << |p|$, i.e., the size of external

constraints of any production in MRG/1 is much smaller than the size of set of evaluation rules of the corresponding RG/1 grammar. So the time taken to resolve constraints in RG/1 sentential forms, i.e., O(|R||p|) is larger than the time taken to resolve the constraints in R during MRG/1 parsing, i.e., O(|R||q|). Since the time taken to resolve the constraints also contributes to the time taken to apply a production. We say that the time taken to apply a production in RG/1 is larger than the time taken to apply a production in MRG/1 based parsing.

As we can see, in all cases MRG/1 based parsing algorithm more efficient than the RG/1 based parsing algorithm. So we can say that, MRG/1 based parsing algorithm is atleast as efficient as, if not better than, RG/1 based parsing algorithm. These results[1] are summarized in Table 5.1.

Parsing Algorithm	Time Complexity			Time taken to apply a production
	General case	$\|V_R\|$=n	When r=n or m=n	
RG/1 based	O(n log n)	$O(n^2)$	$O(n^2)$	more
MRG/1 based	O(n)	$O(n^2)$	$O(n^2)$	less

Table 5.1: Comparision of RG/1 and MRG/1 based Parsing Algorithms.

[1]In this table r and m refers to the number of symbols and number of internal constraints on the r.h.s. of a production.

5.6 Parser Correctness

In this section, we will establish the proof of correctness of the parsing algorithm. We will establish the correctness of the parsing algorithm by the following approach.

1. First we will establish that the parsing algorithm halts in a finite time (after finite number of steps).
2. We will prove that given an MRG/1 grammar and a syntactically incorrect sentence, the parsing algorithm will reject it with an error message.
3. We will prove that given an MRG/1 grammar and a syntactically correct sentence, the parsing algorithm will produce its correct top down parse

Let us prove these points one after the other.

Lemma 5.4: The parsing algorithm will halt in a finite number of steps.

Proof: If we look at the overview of the parsing algorithm in Figure 5.2, we see that the main loop of the parsing algorithm is repeated 'n' times. Within this loop each step takes a constant (finite) time. If an input sentence is syntactically correct then the loop is executed 'n' times. After the termination of the loop the parsing algorithm terminates succesfully. If an input sentence is syntactically wrong then the loop is exited prematurely and the algorithm terminates with an error message. So in all the cases (i.e., whether the input sentence is correct or wrong) the parsing algorithm terminates after a finite number of steps.

Before establishing the other two points let us give some definitions.

Definition 5 *A visual sentence w = (T,RT) is said to be syntactically correct with respect to an MRG/1 grammar G, if w ∈ L(G)*[2].

Definition 6 *A visual sentence w = (T,RT) is said to be syntactically incorrect with respect to an MRG/1 grammar G, if w ∉ L(G).*

Lemma 5.5: Given a MRG/1 grammar for a visual language and a syntactically incorrect sentence w = (T,RT) as input, the parsing algorithm will reject it with an error message.

Proof: We will consider various cases under which a sentence becomes incorrect and how they are detected.

Case 1: When a user constructs a visual sentence using terminal symbols from V_T but do not combine them according to the syntax of the language. For example the following sentence

$\spadesuit^1$ $\clubsuit^1$

$\spadesuit^1$ $\spadesuit^1$

does not belong to L(G) for G given in example 5.2, even though the symbols it uses belongs to V_T. Such errors will be caught by the parsing algorithm during parsing when it searches for the production M(A,a). The entry M(A,a) in such a case will be blank causing the parser to reject the input sentence with error message. In example 5.2, in the first iteration, the parser sees that M(S,♠) is empty so it terminates the parsing process by displaying the error message.

Case 2: When a user constructs a incomplete visual sentence. Let us say a user

[2]The definition of L(G) is given in chapter 4

constructs the following sentence

♣[1]

Even though the above sentence uses terminals from V_T, it is not a valid sentence of L(G), for G given in example 5.2. Such errors will be caught by the parsing algorithm when it tries to expand a nonterminal in the parse tree but finds no input corresponding to it in the input sentence. For the given sentence, at first step the parser applies production 1 of example 5.2 to expand S. After this step when it tries to expand B it finds no input. At this point the parser flashes the error and stops. We think that all the syntactically incorrect sentences fall into one of the above categories, and they are rejected by the parsing algorithm as discussed above.

Lemma 5.6: Given an MRG/1 grammar and a syntactically correct sentence, the parsing algorithm will produce its correct top down parse tree.

Proof: As it is evident by now, our parsing algorithm constructs the parse tree for an input sentence in a top down breadth first manner. Since this is the only type of derivation allowed in MRG/1 (see the description of MRG/1 sentential form in chapter 4), the parse tree produced by our algorithm is correct.

Theorem 5.1: Let G be an MRG/1 grammar and w = (T,RT) be an input sentence. The parsing algorithm will produce a correct parse iff w ϵ L(G).

Proof: Lemmas 5.4, 5.5, 5.6.

Chapter 6

Comparisions

In this chapter we will compare MRG/1 based parsing algorithm with parsing algorithms based on other formalisms. A detailed comparision with other restricted forms of RG grammar, viz RG/1 and 1NS-RG, is also made.

6.1 Comparisions with Other Formalisms

Let us now compare some of the existing grammatical formalisms for Iconic Languages with MRG/1. These formalisms have already been discussed in some detail in chapter 2. Now we will only compare the efficiency of their parsing algorithms with the efficiency of MRG/1 based parsing algorithm. Table 6.1 gives the comparisions.

Grammatical Formalism	Time complexity of Parsing algorithm
Visual Grammars	Exponential
Picture Layout Grammars (restricted form)	Polynomial
Positional Grammars	Exponential
Picture Processing Grammars	Polynomial
MRG/1 Grammars	O(n)

Table 6.1: Comparision of MRG/1 based parsing algorithm with parsing algorithms based on other formalisms.

From the comparisions, it is clear that MRG/1 based parsing algorithm is the most efficient parsing algorithm among all those specified. Since MRG/1 is a restricted form of RG, let us now compare it with other restricted forms of RG viz RG/1 and 1NS-RG.

6.2 Comparision with RG/1

Comparision of MRG/1 and RG/1 as grammatical formalisms was done in chapter 4. It was shown that MRG/1 formalism is superior to RG/1 formalism in terms of clarity of expression. Also it was shown that MRG/1 formalism is equivalent to RG/1 formalism in terms of expressivness. Comparision of MRG/1 and RG/1 based parsing algorithms was done in chapter 5. It was shown that MRG/1 based parsing

algorithm is atleast as efficient as, if not better than, RG/1 based parsing algorithm. These results are summarized in Table 6.2.

Grammatical Formalism	Clarity of Expression	Efficiency of parsing algorithm
RG/1	less	less efficient
MRG/1	more	more efficient

Table 6.2: Comparision of RG/1 and MRG/1 - formalisms and parsing algorithms.

6.3 Comparision with 1NS-RG

Since restrictions imposed on 1NS-RG productions and evaluation rules are completely different from the restrictions imposed on MRG/1 productions, we cannot compare the expressiveness of these two formalisms. We will just compare their parsing algorithms. The parsing algorithm based on 1NS-RG has exponential time complexity in general. The advantage of this parser is that, it takes iconic sentences with no total or partial ordering, and parses them. However it is not a complete parsing algorithm, it is only a recognizer of iconic sentences. It does not generate the parse tree.

The parsing algorithm based on MRG/1 has O(n) time complexity in general. It is a complete parsing algorithm. It generates a parse tree of the iconic sentence, if it is grammatically correct; else prints an error message and stops. Of course the parse tree is produced only in a figurative sense. However the disadvantage of this parsing algorithm is that, it accepts iconic sentences with total ordering. These comparisions between 1NS-RG and MRG/1 are summarized in Table 6.3.

Grammatical Formalism	Efficiency of parsing	Ordering of input sentences	Parse tree is produced
1NS-RG	Exponential	Not required	No
MRG/1	O(n)	Total ordering required	Yes

Table 6.3: Comparision of 1NS-RG and MRG/1 based parsing algorithms.

Chapter 7

Conclusions and Future Research

7.1 Conclusions

Visual Programming Languages are rapidly emerging as a new approach to programming. This is because of their naturalness and ease of programming. But unfortunately, till now, visual programming is not established as a widely used programming approach. This may be because of lack of efficient grammatical formalisms, parsing techniques, and compiler generation tools. Research in visual programming has mainly focussed on VPLs for specific uses. However in mid 80s some work was done on the grammatical formalisms and parsing techniques of VPLs. Grammatical formalisms such as Visual Grammars, Picture Processing Grammars, Picture Layout Grammars, Relation Grammars have been used to model VPLs. Parsing techniques based on these formalisms have also been proposed. But in order to put VPLs on a par with Textual Languages lot more work needs to be done in this area. This research is a step in that direction.

We have analyzed various existing grammatical formalisms and parsing techniques

for a subclass of VPLs called Iconic Languages. Also proposed new grammatical formalism called Modified Relation Grammar/1 (MRG/1 for short) to model Iconic Languages. This is basically a restricted form of Relation Grammar (RG) obtained by restricting the type of productions. It is very much similar to RG/1 grammar, which is another restricted form of RG. Some iconic sentences are modelled using this formalism. Language described by MRG/1 is defined. A detailed comparision with RG/1 is made. It is shown that MRG/1 is superior to RG/1 in terms of clarity of expression. Also it is proved that these two formalisms are equivalent in their expressive power.

A predictive parsing algorithm based on MRG/1 is proposed. The parsing algorithm is applied to two examples and all the steps are traced. This clarifies the detailed working of the algorithm. The time complexity of the parsing algorithm is derived, which is found to be linear. A detailed comparision with RG/1 based parsing algorithm is made. It is shown that MRG/1 based parsing algorithm is at least as efficient as, if not better than, RG/1 based parsing algorithm. Proof of correctness of the MRG/1 based parsing algorithm is also given.

The time complexity of MRG/1 based parsing algorithm is compared with the time complexities of parsing algorithms based on other formalisms such as Visual Grammars, Picture Layout Grammars, Positional Grammars, and Picture Processing Grammars. It is shown that MRG/1 based parsing algorithm is relatively more efficient. The time complexity of MRG/1 based parsing algorithm is also compared with the time complexity of 1NS-RG based parsing algorithm.

From all the comparisions we conclude that MRG/1 is a clear of way of expressing VPL syntax, and parsing based on it is more efficient.

7.2 Future Work

In this work we have assumed that the visual sentence is preprocessed by the lexical analyzer. The lexical analyzer converts the input picture into a totally ordered mutiset of visual icons (tokens), with spatial relations among those icons. This representation of the input is taken by the parser and parsed. The specification and implementation of such a lexical analyzer can be considered as future work. Such a lexical analyzer can be implemented in two ways.

- The lexical analyzer can construct the graphic function sequences corresponding to the input icons and spatial relations, as the visual sentence is being generated. These graphic function sequences can then be sorted based on spatial relations to generate the totally ordered sentence.

- The lexical analyzer can take the postscript file of the input picture and then scan it to deduce the spatial relations among the icons. The sentence can be simultaneously ordered in this case.

Throughout this work we have assumed that the parsing algorithm produces figurative parse tree. In other words it prints only the sequence of applied productions. We can associate semantic attributes with our formalism and enhance our parsing algorithm to carry out semantic actions associated with the applied productions. This can also be regarded as extension to this work.

Finally all the three phases of the compiler, viz lexical analysis, syntax analysis, and semantic analysis, can be put together and a complete compiler can be built for Iconic Languages.

Bibliography

[1] A.V. Aho. and J.D. Ullman. *Principles of Compiler Design.* Addison-Wesley, third edition, 1979.

[2] Herot C. F. Kramlich D. A. Brown G. P., Carling R. T. and P. Souza. Program Visualization : Graphical Support for Software Development. *IEEE Computer*, 18(8):27–35, Aug 1985.

[3] Pong M. C. and Ng N. PIGS - a System for Programming with Interactive Graphical Support. *Software - Practice and Experience*, 13(9):847–855, Sept 1983.

[4] Shu N. C. FORMAL : A Forms-Oriented and Visual-Directed Application System. *IEEE Computer*, 18(8):38–49, Aug 1985.

[5] Smith D. C. *Pygmalion : A Creative Programming Environment.* PhD thesis, Department of Computer Science, Stanford University, 1975.

[6] G. Nota. G. Pacini. G. Tortora. M. Tucci. C. Crimi., A. Guerico. Relation Grammars for Modelling Multi-dimensional Structures. *IEEE Workshop on Visual Languages*, 1990.

[7] S. K. Chang. Picture Processing Grammar and its Applications. *Information Sciences*, 3:121–148, 1971.

[8] Shi-Kuo Chang. Ten Years of Visual Languages Research. *IEEE Symposium On Visual Languages*, 1994.

[9] Gennaro Costagliola. and Shi-Kuo Chang. DR PARSERS : A Generalization of LR parsers. *IEEE Workshop on Visual Languages*, 1990.

[10] Jay Earley. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM*, 13, 1970.

[11] Herot C. F. Spatial Management of Data. *ACM Transactions on Database Systems*, 5(4):493–514, Dec 1980.

[12] Rochart J. F. and Flannery L. S. The Management of End User Computing. *Communicatios of the ACM*, 26(10):776–784, Oct 1983.

[13] G. Tortora. M. Tucci. F. Ferrucci., G. Pacini. and G. Vitiello. Efficient Parsing of Multi-dimensional Structures. *IEEE Workshop on Visual Languages*, 1991.

[14] Jerome Feder. Plex Languages. *Information Sciences*, 3(3):225–241, 1971.

[15] K. S. Fu. Tree Languages and Syntactic Pattern Recognition. In C. H. Chen, editor, *Pattern Recognition and Artificial Intelligence*, pages 257–291. Academic Press, 1976.

[16] K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, 1982.

[17] G. Vitiello G. Costagliola., M. Tucci. Parsing Nonlinear Languages. *IEEE Transactions on Software Engineering*, 20(9), 1994.

[18] Masaru Tomita. Gennaro Costagliola. and Shi-Kuo Chang. A Generalized Parser for 2D languages. *IEEE Workshop on Visual Languages*, 1991.

[19] Eric J. Golin. *A method for the Specification and Parsing of Visual Languages.* PhD thesis, Brown University, 1991.

[20] Yoshimoto I. Tanaka M. Hirakawa M., Monden N. and Ichakawa T. HI-VISUAL : A Language Supporting Visual Interaction in Programming. In S. K. Chang et al., editor, *Visual Languages*, pages 233–259. Plenum Press, 1989.

[21] L. Weitzman K. Wittenburg and J. Talley. Unification-Based Grammars and Tabular Parsing for Graphical Languages. *Journal of Visual Languages and computing*, 2, 1991.

[22] Fred Lakin. Spatial Parsing for Visual Languages. In Tadao Ichikawa. Shi-Kuo Chang. and Panos S. Ligomenides, editors, *Visual Languages*, pages 35–85. Plenum Press, 1989.

[23] G. Tortora. M. Tucci., F. Ferrucci. and G. Vitiello. A Predictive Parser for Visual Languages Specified by Relation Grammars. *Proceedings IEEE Symposium on Visual Languages*, 1994.

[24] Gerald Masini. and Roger Mohr. MIRABELLE, a System for Structural Analysis of Drawings. *Pattern Recognition*, 16(4):363–372, 1983.

[25] Glinert E. P. and Tanimoto S. L. Pict : An Interactive Graphical Programming Environment. *IEEE Computer*, 17(11):7–25, Nov 1984.

[26] Reiss S. P. PECAN : Program Developement Systems that Support Multiple Views. *IEEE Transactions on Software Engineering*, 11(3):276–285, March 1985.

[27] T. Ichikawa. P. A. Ligomenides., S. K. Chang. *Visual Languages*. Plenum Press, Newyork, 1989.

[28] Maragaret A. Korfhage Robert R. Korfhage. Criteria for Iconic Languages. In S. K. Chang et al, editor, *Visual Languages*, pages 207–231. Plenum Press, 1989.

[29] Bing Yu. S. K. Chang., M. J. Tauber. and Jing-Sheng Yu. A Visual Language Compiler. *IEEE Transactions on Software Engineering*, 1989.

[30] Linda G. Shapiro. and Robert J. Baron. ESP^3 : A Language for Pattern Description and a System for Pattern Recognition. *IEEE Transactions on Software Engineering*, 3(2):169–183, March 1977.

[31] Alan C. Shaw. A Formal Picture Description Scheme as a Basis for Picture Processing Systems. *Information and Control*, 14:9–52, 1969.

[32] Nan C. Shu. *Visual Programming*. Van Nostrand ReinHold Company, Newyork, 1988.

[33] K. Wittenburg. Earley-style Parsing for Relational Grammars. *Proceedings of IEEE Workshop on Visual Languages*, 1992.

Vita

- Mohammed Ather Ahmed.
- Born on Febrary 10, 1971, at Hyderabad, India.
- Received Bachelor of Engineering degree in Computer Science and Engineering from Osmania University, Hyderabad, India, in 1992.
- Joined the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, as a Research Assistant, in September 1993.
- Completed Master of Science degree in Information and Computer Science from KFUPM, Dhahran, Saudi Arabia, in December 1996.